SOPC 综合实验系统

实验指导书

济南大学控制学院

责任编辑：济南大学信息学院电子信息中心

目录

目录 1

第一章 系统的硬件单元及使用说明 4

一、本实验箱的结构特点 4

1、系统构成 4

2、创新设计应用领域 4

3、适用专业范围 5

二、各模块电路说明 5

1、数码管显示模块 5
2、AD 转换模块 6
3、DA 转换模块 6
4、以太网模块 7
5、VGA接口模块 7
6、PS/2 接口模块 8
7、串行接口模块 8
8、开关量输入输出模块 8
9、按键模块 9
10、LED 灯指示模块 10
11、音频接口模块 10
12、TFT液晶接口模块 11
13、USB接口电路 11

第二章 QUARTUS II和NIOS II 软件的使用方法 12

一、QUARTUS II软件的使用方法 12
二、SOPC BUILDER / NIOS II IDE软件使用方法 21

第三章 VHDL基本程序设计实验 29

实验一 组合逻辑半加器的设计 29
实验二 使用VHDL设计组合逻辑全加器 36
实验三 带进位输入的 8 位加法器 41
实验四 全减器 46
实验五 四位向量加法/减法器 50
实验六 组合逻辑 3-8 译码器的设计 53
实验七 数据比较器 57
实验八 多路数据选择器 60
实验九 编码器 64
实验十 计数器 69
实验十一 7 段数码管控制接口 72
实验十二 16*16 点阵设计 76
实验十三 MOORE机 83
实验十四 MEALY机 87

实验十五 D/A接口 91
实验十六 A/D接口 94
实验十七 动态数码管显示实验 98
实验十八 频率信号源实验 102

第四章 SOPC系统基本实验 109

实验一 Hello-Led流水灯实验 109
实验二 数码管显示实验 117
实验三 按键输入中断实验 121
实验四 定时计数器实验 125
实验五 串行口通信实验 129
实验六 存储器配置实验 132
实验七 4 乘 4 键盘实验 138

第五章 SOPC系统综合实验 141

实验一 高速DAC实验 141
实验二 DDS实验 149
实验三 高速ADC实验 155
实验四 静态数码管显示实验 158
实验五 VGA彩条显示实验 160
实验六 PS2 键盘实验 166
实验七 USB数据读写实验 171
实验八 TFT真彩屏实验 173
实验九 SD卡实验 176
实验十 UC\OS-II操作系统移植实验 181
实验十一 PS2 鼠标控制实验 188
实验十二 音频接口实验 191
实验十三 百兆以太网实验 199
实验十四 四相步进电机实验 208
实验十五 CAN总线实验 211

第一章　系统的硬件单元及使用说明

一、本实验箱的结构特点

1、系统构成

SOPC 综合实验系统开发实验平台采用 **ALTERA Cyclone III EP3C16 处理器**，集众多功能于一体，采用全新的“主控制板+……+核心板+平台主板”自由组合式结构。根据实验研发需求，可实现基于 **ALTERA**、**XILINX**、**ACTEL**、**Lattice** 等厂家的软核/硬核处理器相结合的嵌入式系统设计。本实验系统实验平台功能强大，硬件接口丰富，平台嵌入式软件除支持 C、C++语言开发之外，还支持 uC/OS II、uClinux 嵌入式操作系统。独特的核心板扩展接口可完美实现核心板与核心板、核心板与平台主板之间无缝连接，实现复杂的系统功能。用户可根据需求，选择 SOPC 和 CPLD 组合、单片机和 SOPC 的组合……通过模式转换功能，不同的硬件模块间，可实现多种独立的具有特色的综合模块组合。平台优异的综合性设计大大提高了开发平台的利用率，是 SOPC、EDA、SOC、单片机相互结合的实验教学、电子系统设计创新实验室、嵌入式系统实验室、科研开发最理想的选择，同时也是各高等院校参加电子设计大赛最理想的创新开发训练平台。

本研究中心一直致力于技术创新，不遗余力地支持企事业单位以及高校教育事业的发展并推进其科研机构研发能力的提升。为了满足不同的设计需求，一切从设计出发，综合创新开发实验平台目前已配套开发出近 20 种扩展核心开发板。核心开发板自成系统，功能强大，实用性强，可采用如下三种方式完成系统设计开发：

主控制板自成嵌入式系统，所提供的硬件资源丰富，标准工业级多层板设计，可脱离实验箱硬件平台，独立开发使用。

核心板与主控制板实验箱平台配套开发使用，除了提供更为丰富的硬件平台接口资源以外，还为用户提供可扩展的自由定制的特色模块，灵活性强，可实现更多、更强、更有创意的综合系统功能。

主控制芯片与主控制板上下叠加结合使用，国内独创，根据国际嵌入式技术的发展可以随时更换主流的控制芯片，可完美实现各种电子系统板级创新设计构想，拓宽嵌入式软硬件设计视野，训练出一流的高科技人才。

2、创新设计应用领域

基于 CPLD/FPGA 和 VHDL/Verilog 的普通和典型 EDA 实验与开发；

基于 SOPC 的 Nios II 嵌入式系统实验与开发；

经典 8051/89C51 单片机 IP 核开发应用；

普通单片机与 FPGA 综合实验与开发；

现代计算机组成原理实验；

Altera SOPC 与 DSP 的高级创新应用开发；

基于 ALTERA Nios II 多核 32 位处理器的创新应用开发；

ALTERA Nios II 核心板与 Nios II 核心板之间的创新开发应用；

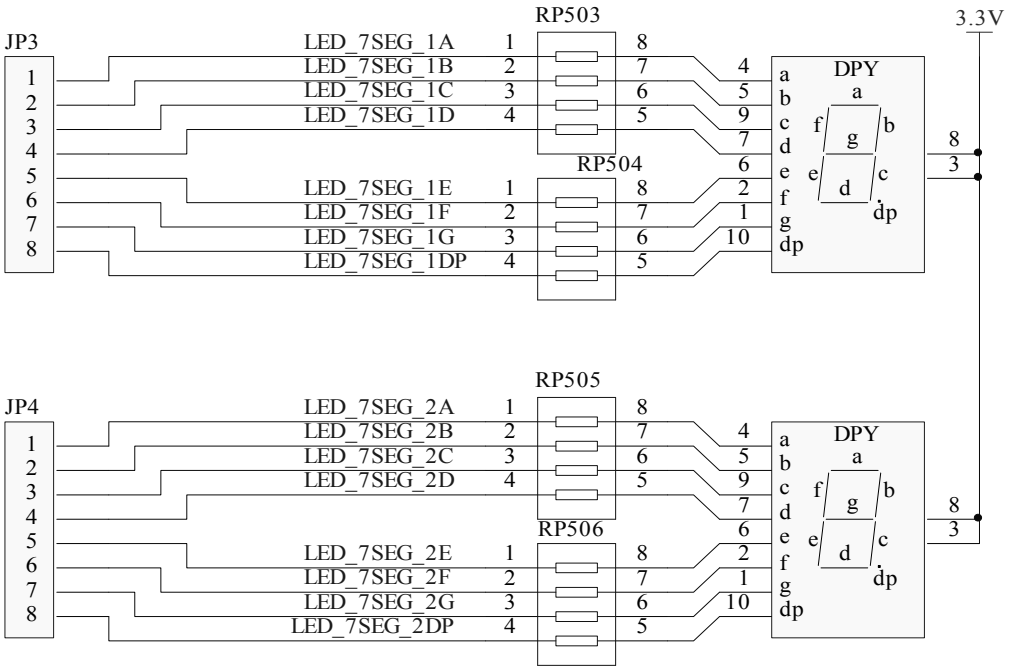
ALTERA Nios II 核心板与 Xilinx Microblaze 核心板之间的并行创新开发应用；

3、适用专业范围

本系统综合创新开发实验平台专为电子系统级产品设计、嵌入式软硬件设计、EDA 基础教学、IP Core 开发与验证（包含 8/16/32 位 CPU 处理器设计）、DSP 图像/通讯创新开发设计……适用于计算机和电子类等专业本科生、研究生、博士生及全国相关各科研院所、企业单位，如计算机科学、微电子、音视频与多媒体教学、现代计算机组成原理、通信、信息技术与仪器仪表、电子工程、机电一体化、自动化等相关专业。

二、各模块电路说明

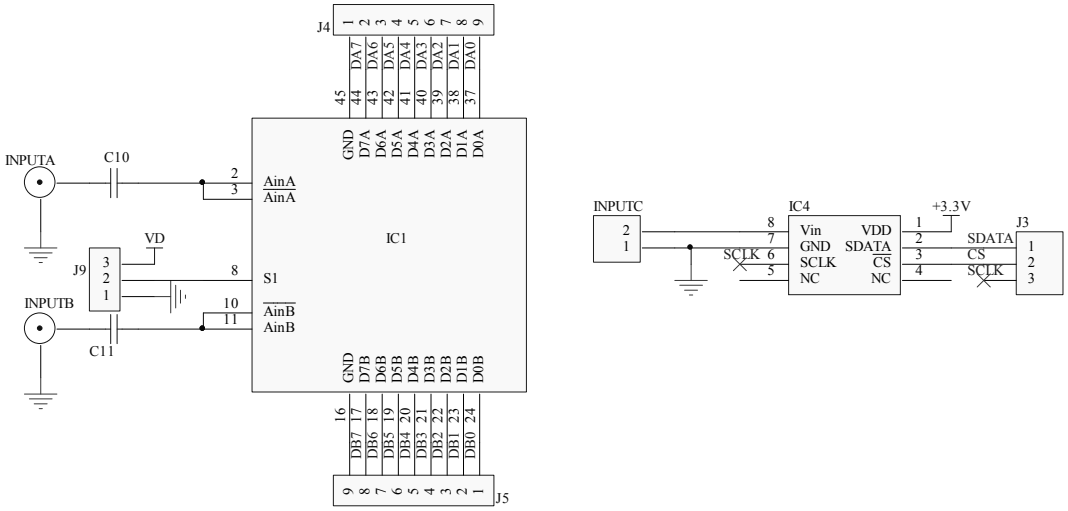
1、数码管显示模块



模块说明：

数码管的断信号由 FPGA 直接驱动，JP3，JP4 代表两个共阳极数码管的 A、B、C、D、E、F、Dp 段；数据 0、1、2、3、4、5、6、7、8、9 对应的断码分别为：0xC0、0XF9、0xA4、0xB0、0x99、0x92、0x82、0xF8、0x80、0x90 。

2、AD 转换模块



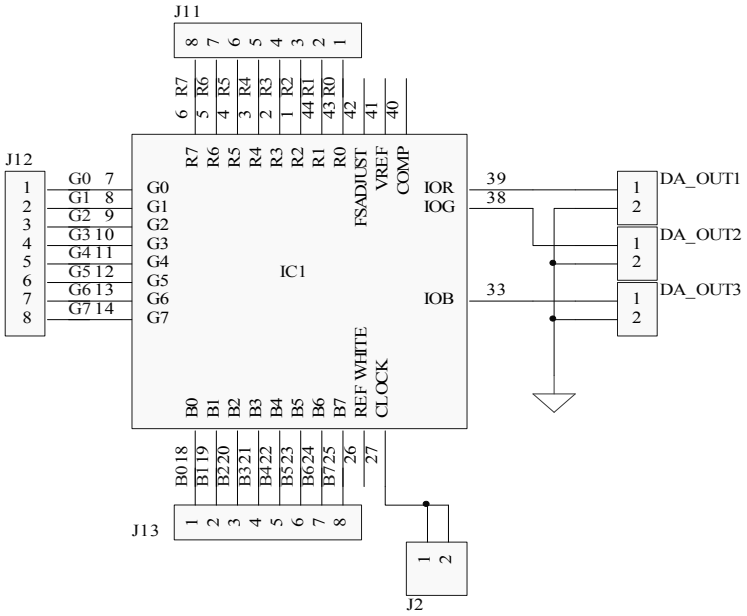
模块说明：

此模块设计了两种 AD 转换器件：

AD9288 是采用了并行双通道独立 8 位、高速采样（100MHZ）的 A/D 器件，模拟信号分别通过 INPUT_A、INPUT_B 输入，时钟输入采用 FPGA 控制的 10-100MHZ 时钟信号，数据采用 8 位并行输出。FPGA 控制采样率，此实验可以很快的验证采样定律。注意在使用该模块的过程中应该将入信号应该为调节到 0 到 1V 的电压范围内的高频交流信号。

AD7920 是 12 位高速串行 AD，时钟输入信号 0—3M，电压输入为 0—3.3v 范围内直流电压。

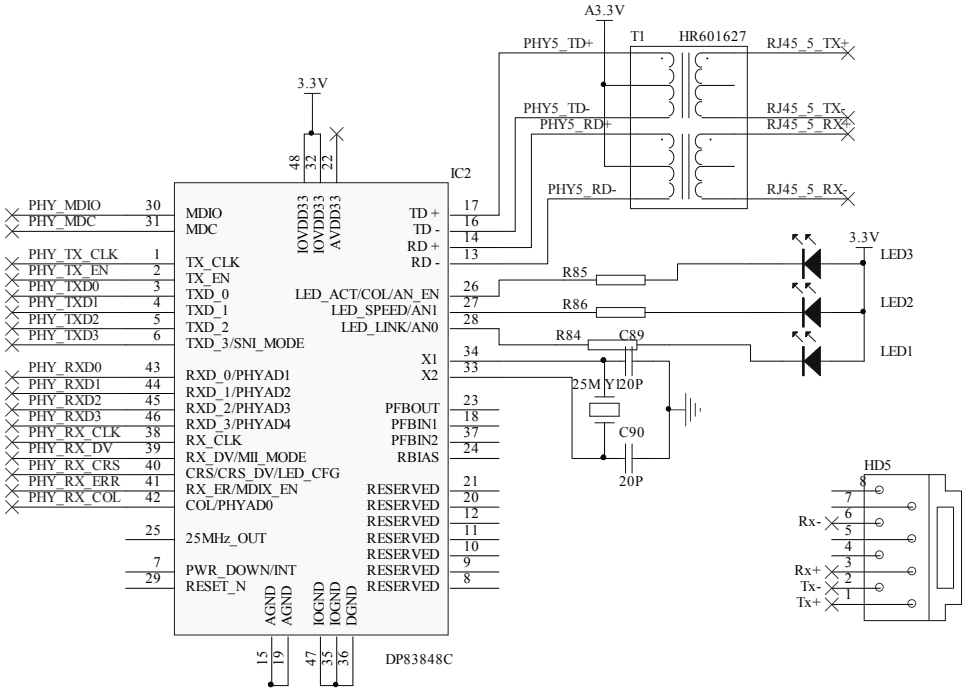
3、DA 转换模块



模块说明：

ADV7120 是美国 ADI 公司出品的高速数模转换电路，在单芯片上集成了 3 个独立的 8 位高速 D/A 转换器，特别是用于高速 D/A 转换的应用系统。J11、J12、J13 作为 FPGA 数据输入端口分别对应 DA_OUT_1、DA_OUT_2、DA_OUT_3 三路模拟电压输出。

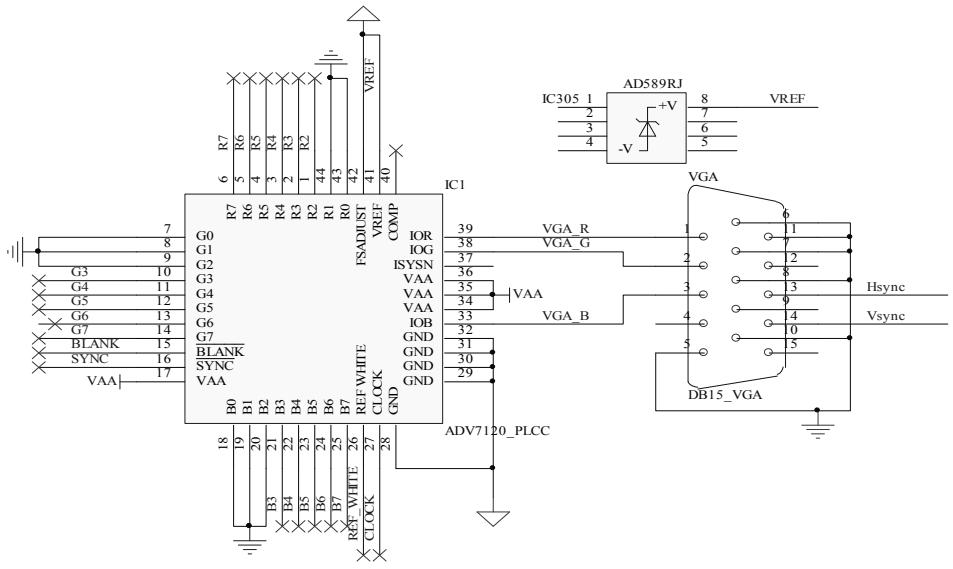
4、以太网模块



模块说明：

该模块为百兆以太网设计模块，FPGA 通过排线连接 J2、J3、J4 对以太网数据进行读写和控制。

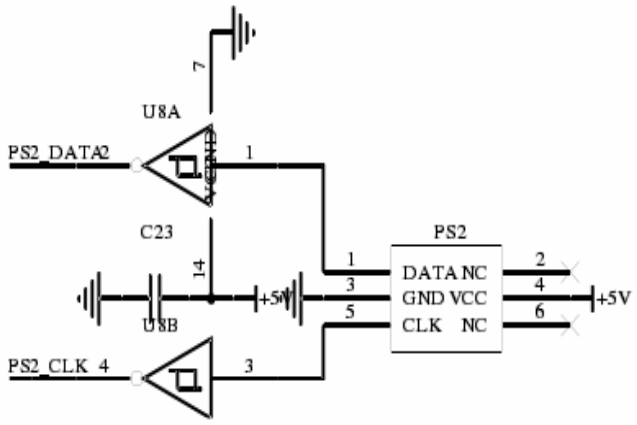
5、VGA接口模块



模块说明：

上电时 CPLD 内部逻辑会自动加载存储器信息控制 VGA 显示时序和图像，实验者，只需要由 FPGA 通过数据线和地址线对存储器单元进行操作。

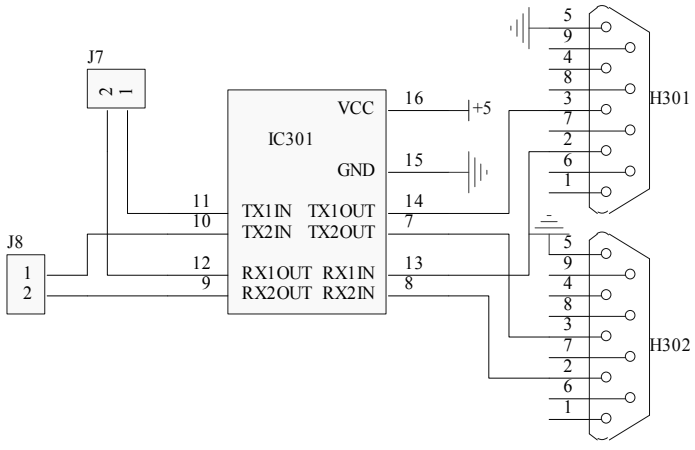
6、PS/2 接口模块



模块说明：

该模块设计有两个 PS/2 接口，都可以接 PS/2 设备，其时钟线和数据线通过排线与 FPGA 相连。

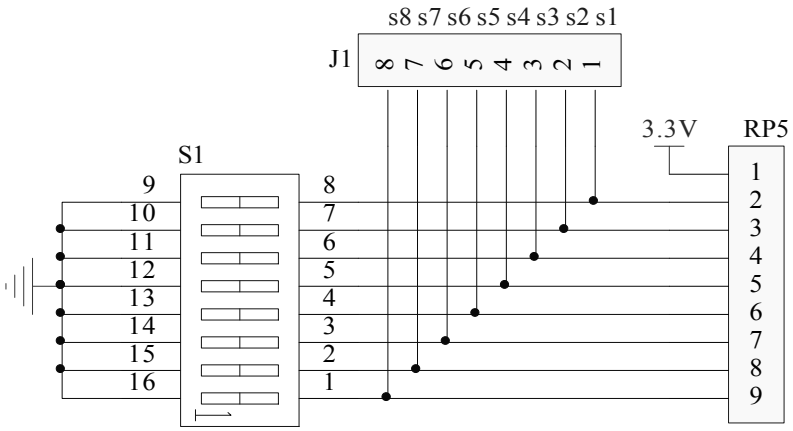
7、串行接口模块



模块说明：

该模块设计了两个 RS232 串行接口，可以用于 FPGA 与其他设备进行数据通信。其中 J7 与 H301 为 COM1 输入输出口，J8 与 H302 为 COM2 舒入输出口。

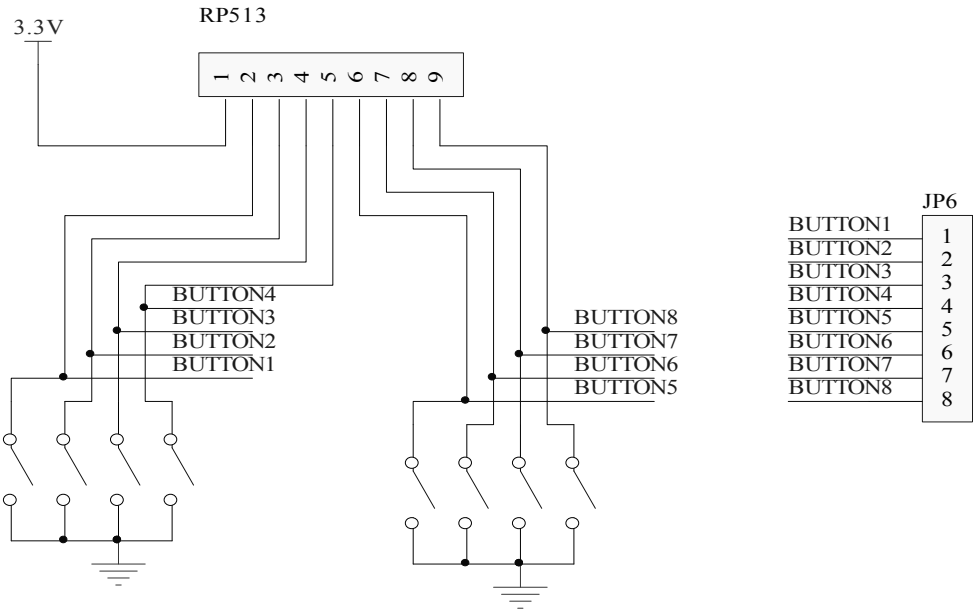
8、开关量输入输出模块



模块说明：

S1—s8 是带自锁的单刀单执拨码开关，在开关未拨动时是低电平，拨动时 J1 为高电平并且保持高电平不变，只有回拨开关时 J1 才恢复低电平输入。J1 通过排线与其他模块相连接。

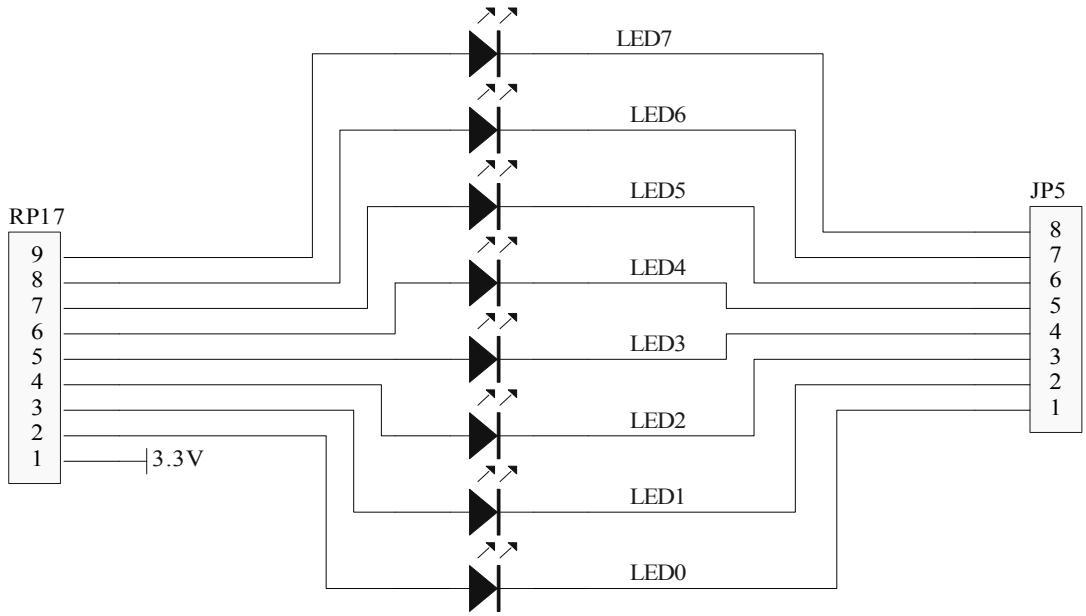
9、按键模块



模块说明：

此模块共有 8 个按键，BUTTON1—BUTTON8 是轻触按键；在按键未按下时 JP6 为高电平输入，按键按下后 JP6 对 FPGA 输入低电平，松开按键后恢复高电平输入。JP6 通过排线与 FPGA 相连接。

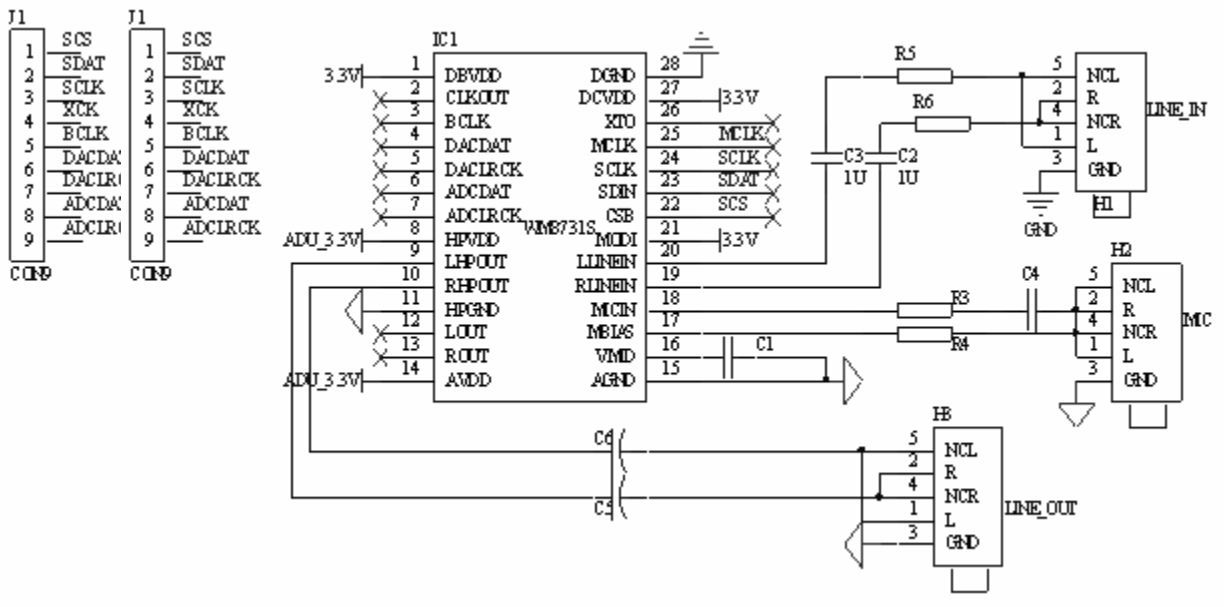
10、LED 灯指示模块



模块说明：

该模块有 8 个 LED 指示灯，在使用时候只需要用排线连接 JP5 和 FPGA 连接，FPGA 输出低电平时指示灯亮。

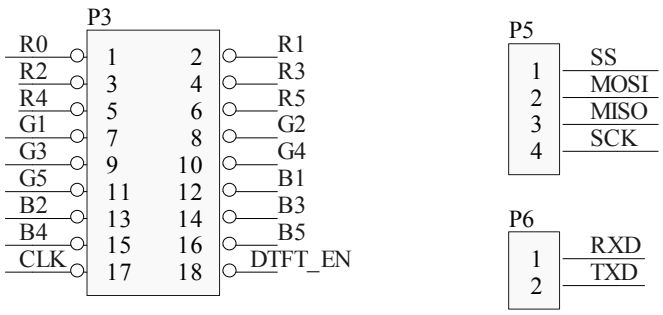
11、音频接口模块



模块说明：

J1 通过排线与 FPGA 相连接处理音频数据，H1 接口为左右声道输出，H2 为 MIC 输入。

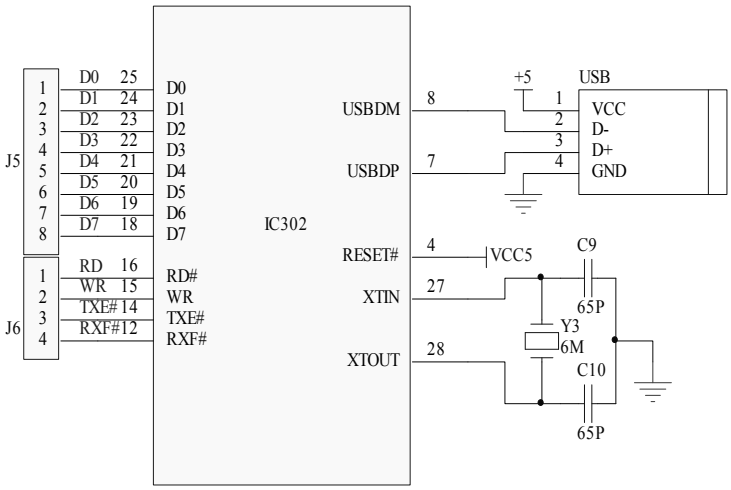
12、TFT液晶接口模块



模块说明：

本模块为真彩触摸屏显示实验，P3 为 TFT 控制接口，P5 和 P6 接口为触摸信号输出模式分别为 SPI 模式和串口模式。

13、USB接口电路



模块说明：

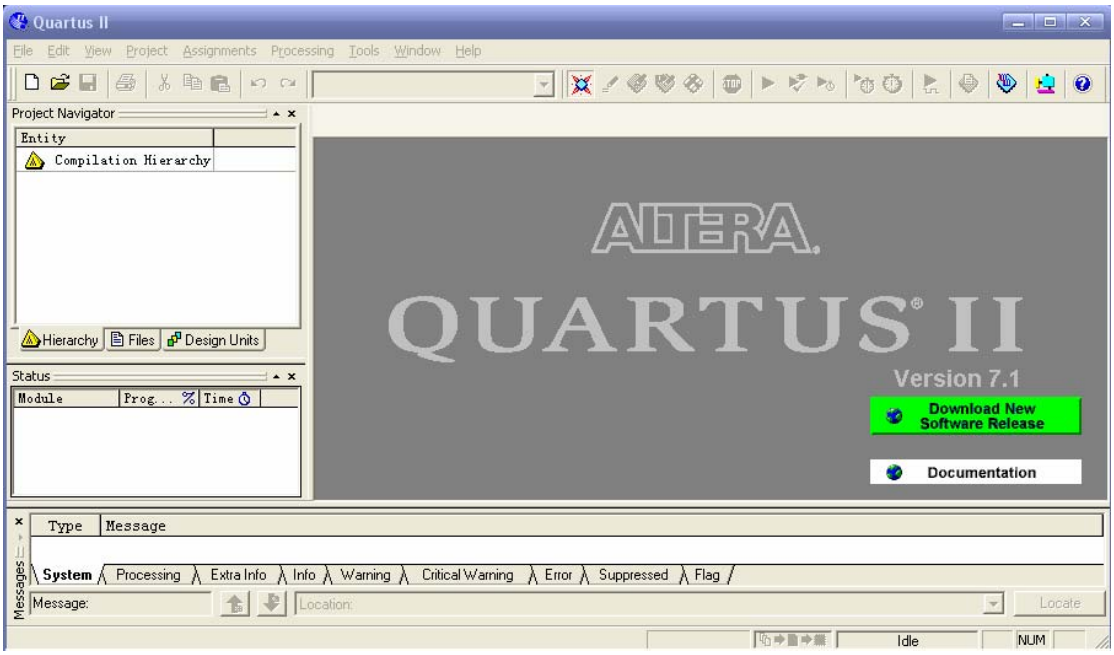
本模块中 J5，J6 通过排线与 FPGA 相连，其中 J5 为 USB 数据，J6 为控制线。

第二章 Quartus II和Nios II 软件的使用方法

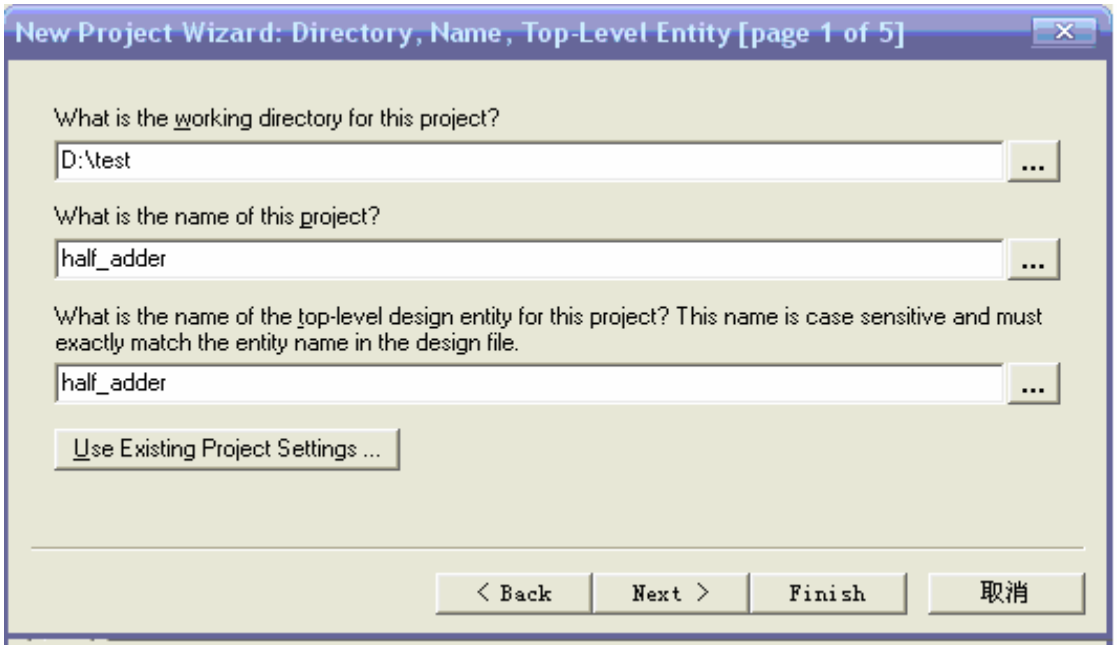
一、Quartus II软件的使用方法

1、创建工程

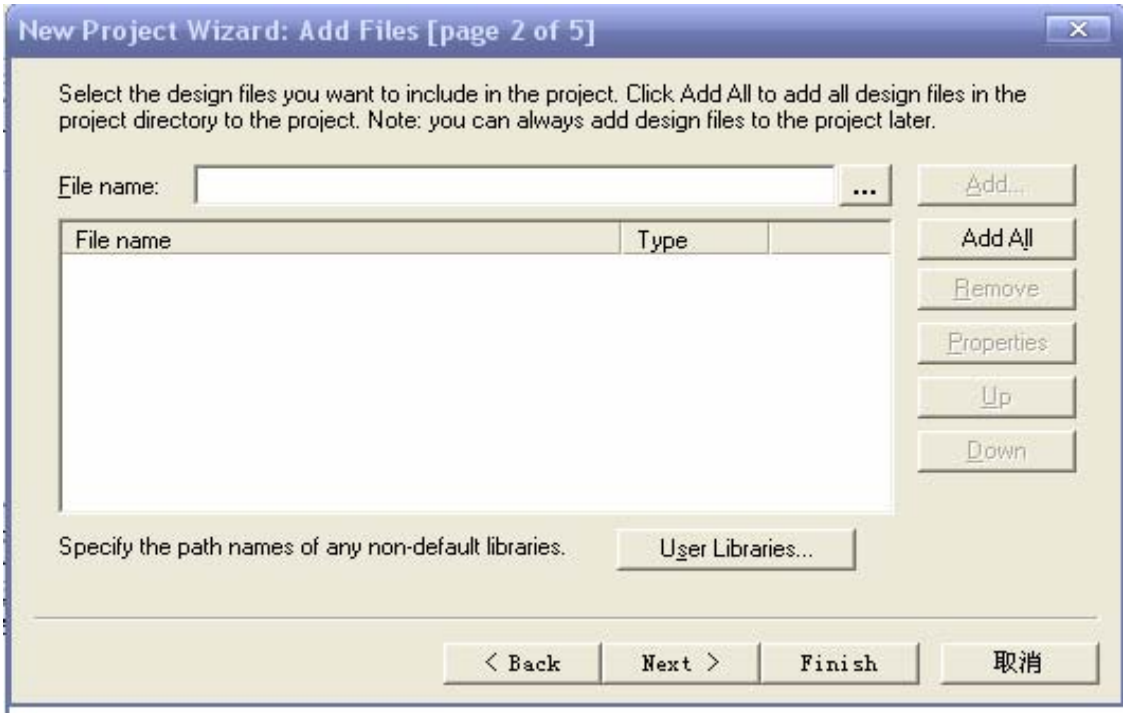
（1）运行 quartus II 软件，如下图：



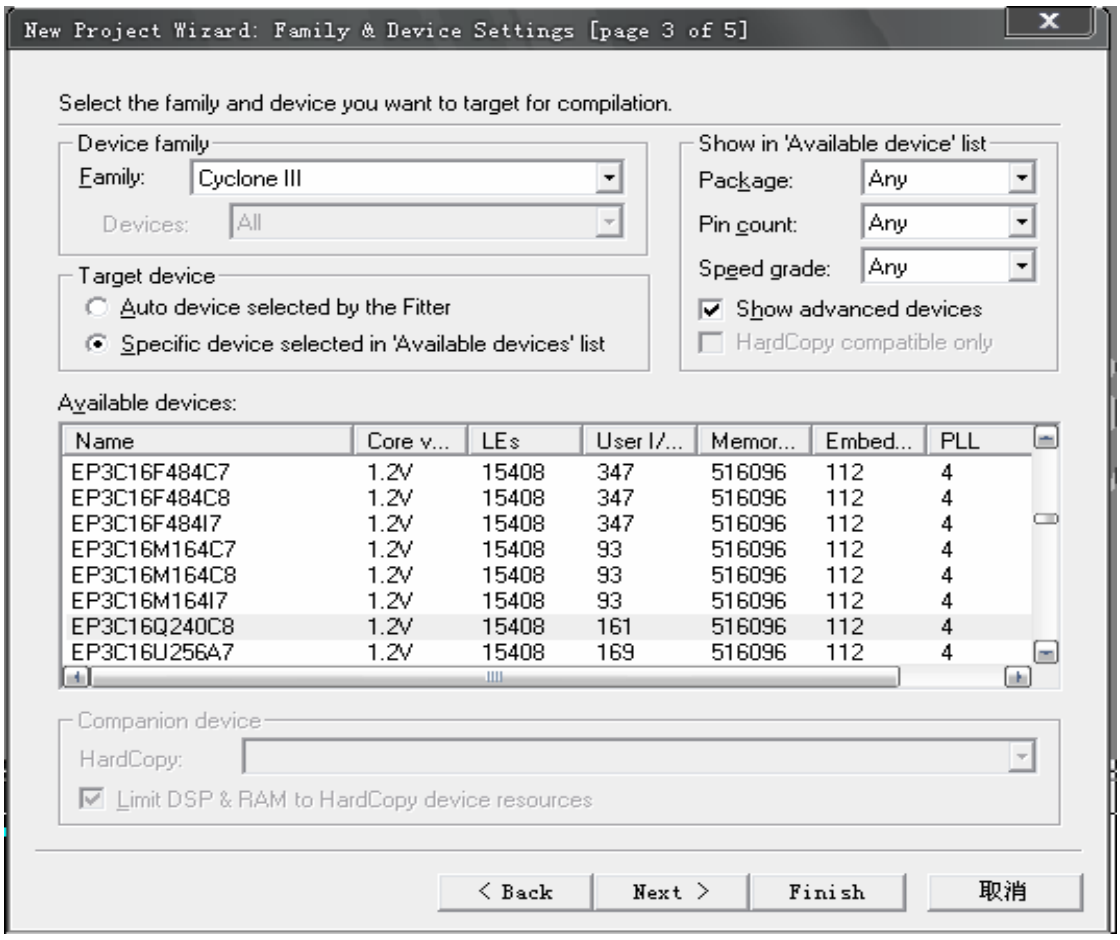
（2）建立工程，File→New Project Wizad，既弹出“工程设置”对话框，如下图：



（3）单击此对话框最上一栏右侧的“...”按钮，在 D 盘建一个工程文件夹，取名 test。单击“打开”，在第二行和第三行中填写为“half_adder”，按 Next 按钮，出现添加工程文件的对话框，如下图：

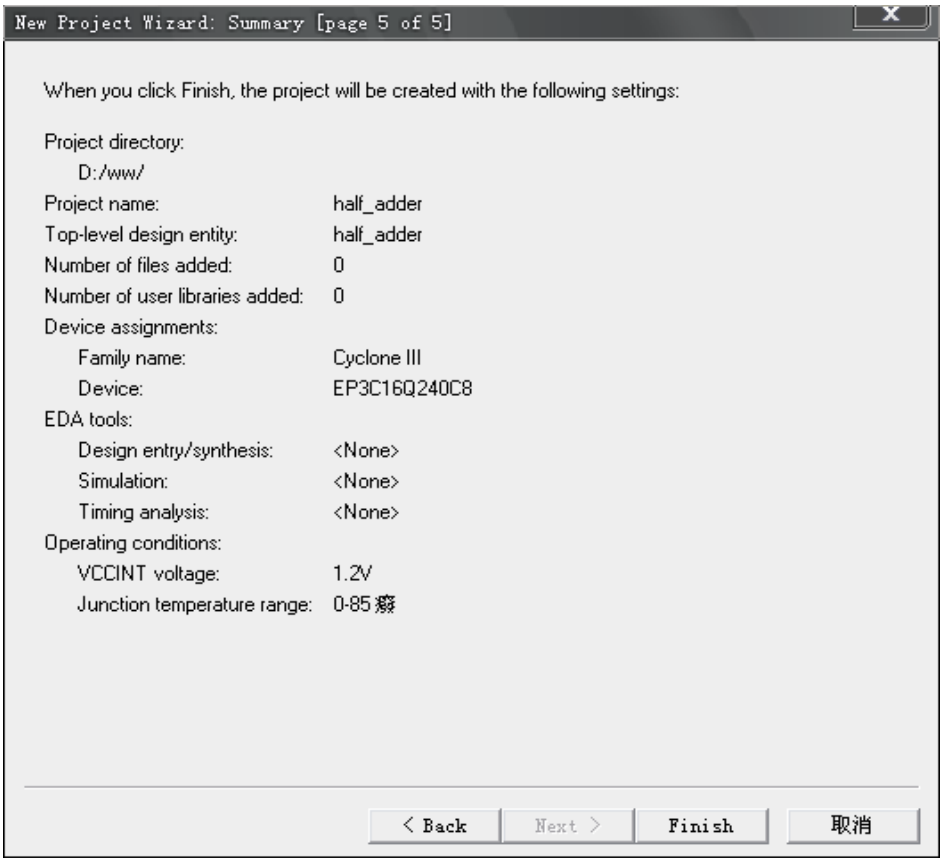


这里我们先不管它，直接按 next 进行下一步，选择 FPGA 器件的型号，如下图：



（4）在 Family 下拉框中，我们选择 CycloneIII 系列 FPGA，选择此系列的具体芯片 EP3C16Q240C8。

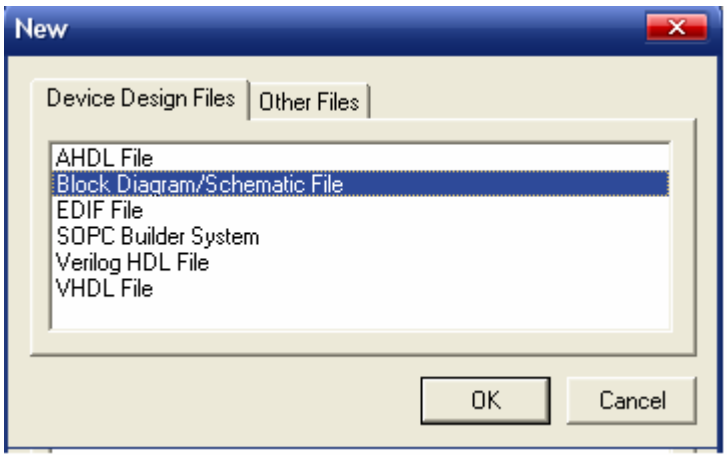
执行 next 出现选择其它 EDA 工具对话框，我们用 Quartus II 的集成环境进行开发，因此这里不作任何改动，按 next 进入工程的信息总概对话框：



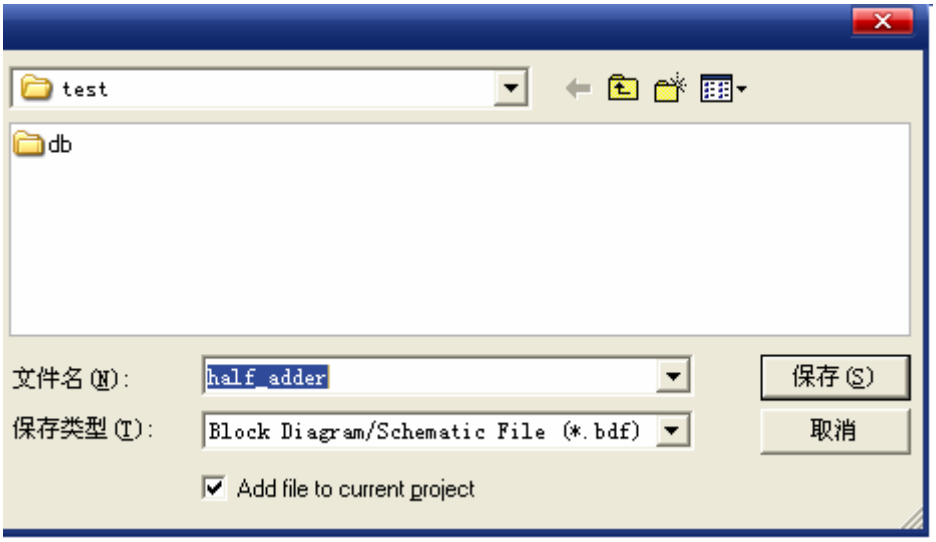
（5）按 Finish 按钮即建立一个项目。

2、建立顶层图

（1）执行 File→New，弹出新建文件对话框，如下图：

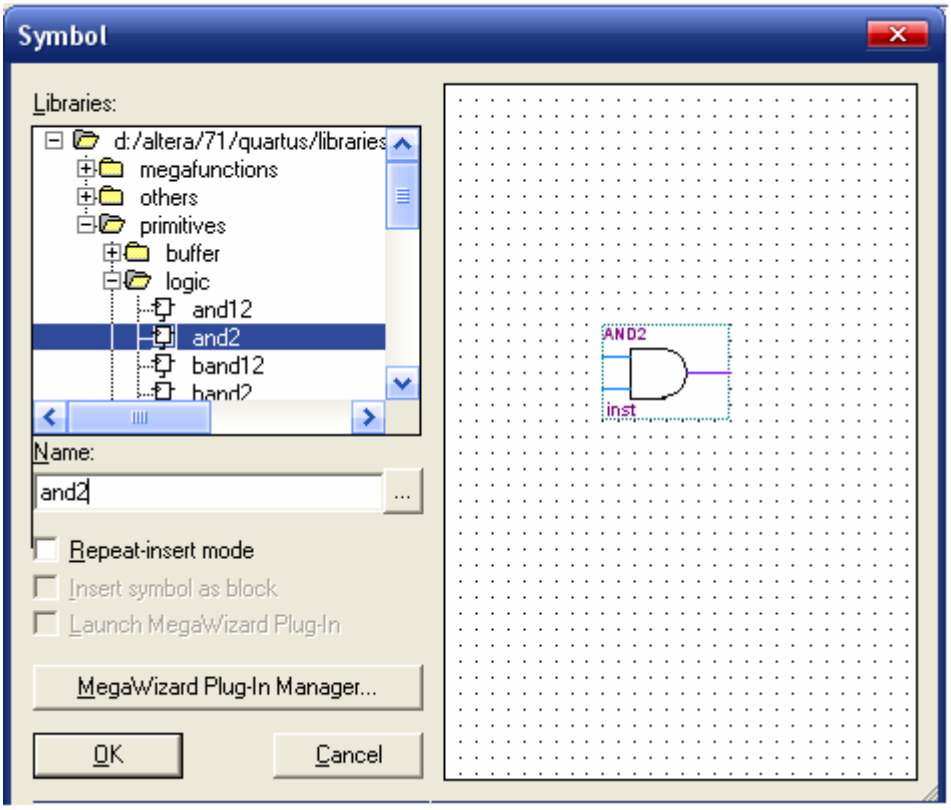


选择“Block Diagram Schematic File”按 OK 即建立一个空的顶层图，缺省名为“Block1.bdf”，我们把它另存为（File→Save as）,接受默认的文件名，并将“Create new project based on this file”选项选上，以使该文件添加到工程中去，如下图：



（2）添加逻辑元件（Symbol）

双击顶层图图纸的空白处，弹出添加元件的对话框，如下图：

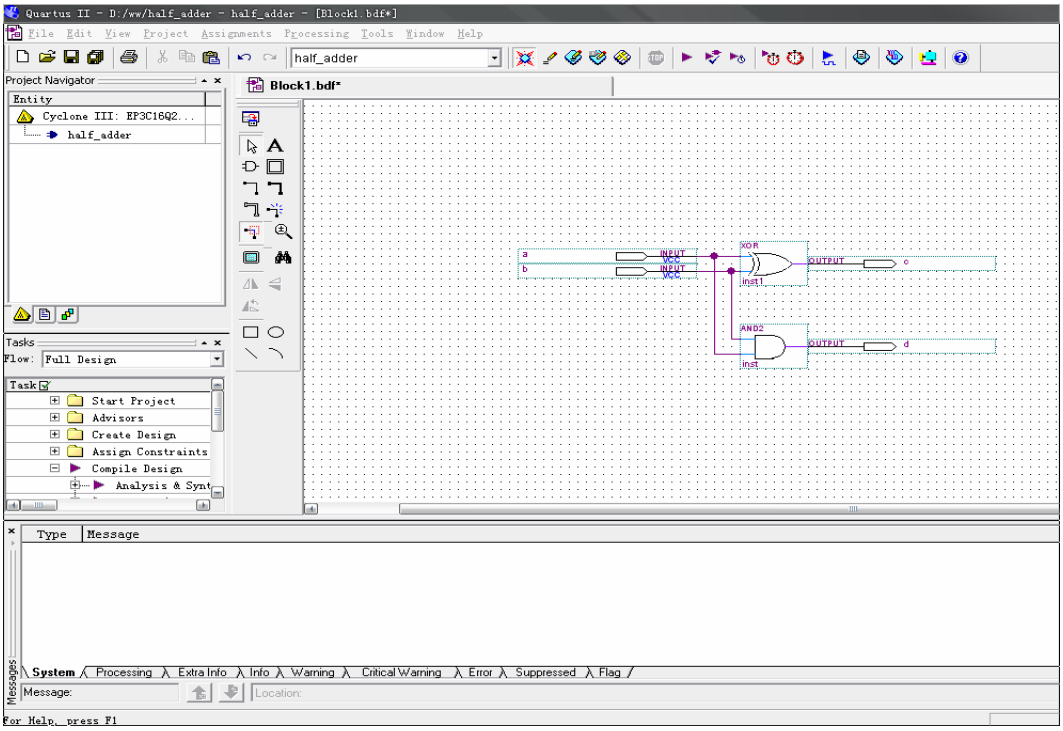


在 libraries 里寻找所需要的逻辑元件，如果知道逻辑元件的名称的话，也可以直接在 Name 一栏敲入名字，右边的预览图即可显示元件的外观，按 OK 后鼠标旁边即拖着一个元件符号，在图纸上点击左键，元件即安放在图纸上。在图纸上分别安放异或门（xor），与门(and2)，输入（input），输出（output）。

（3）连线

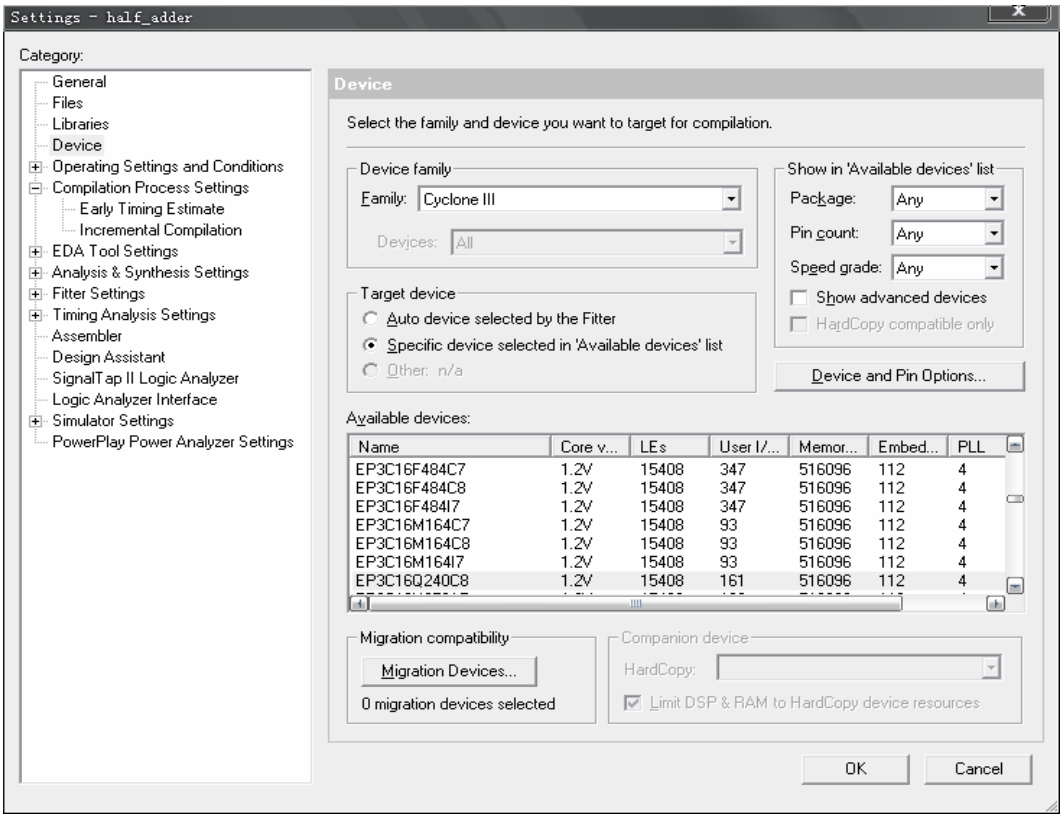
将鼠标移到 symbol 连线端口上，鼠标变成图示形状，按下左键拖动鼠标到另一个 symbol 的

连线端。双击输入输出符号名字“pin_name”，将它们修改为 a，b，s，co，如下图：



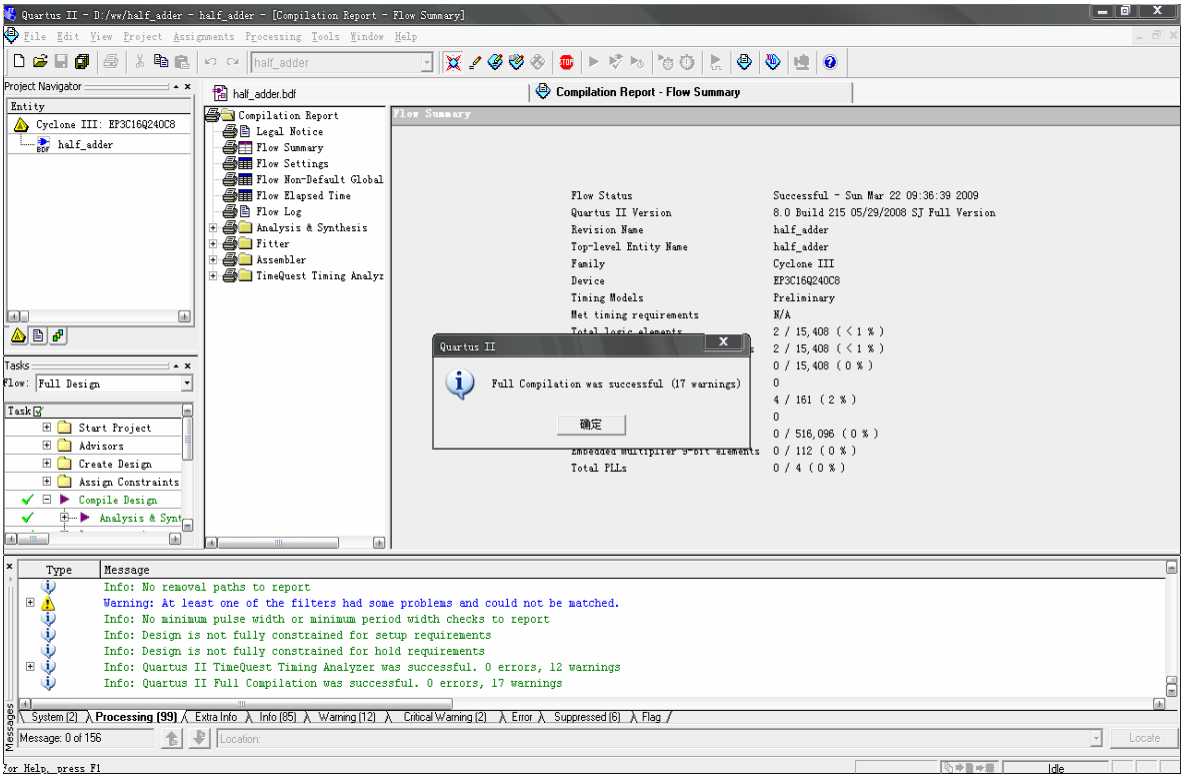
（4）设置

在建立工程时我们选定了芯片型号，也可以在这一步设定，在菜单 Assignments→Device，弹出对话框，如下图：



（5）编译

按主工具栏上的编译按钮即开始编译，Message 窗口会显示一些编译信息，最后编译成功弹出提示，如下图：

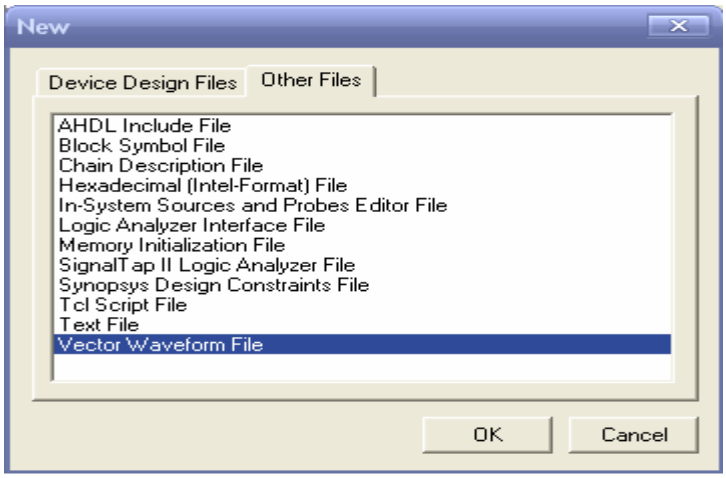


3、仿真

对工程编译通过后，必须对其功能和时序性质进行仿真测试，以了解设计结果是否满足原设计要求。具体步骤如下：

（1）时序仿真

打开波形编辑器。选择菜单 File 中的 New 项，在 New 窗口中选择 Other Files 中的 Vector Waveform File 项，如下图所示：



单击 OK 按钮，即出现空白的波形编辑器，设置仿真时间区域。对于时序仿真来说，将仿真时间轴设置在一个合理的时间区域上十分重要。通常设置的时间范围在数十微秒间。

在 Edit 菜单中选择 End Time 项，在弹出的窗口中的 Time 栏处输入 50，单位选择“us”，整个仿真域的时间即设定为 50us，单击 OK 按钮，结束设置。

（2）波形文件存盘

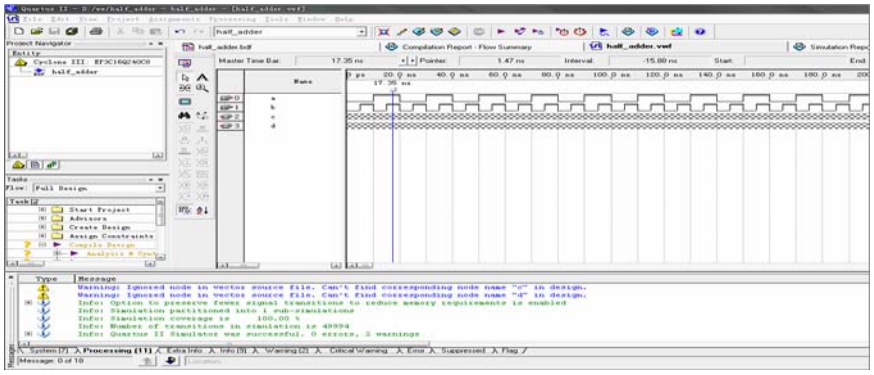
选择 File 中的 Save As 项，将以默认名的波形文件存入文件夹中。

（3）将工程的端口信号节点选入波形编辑器中

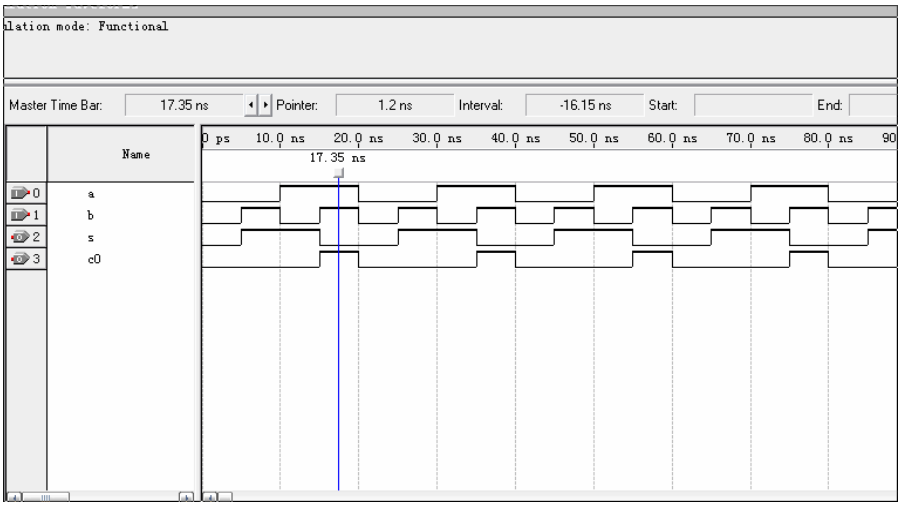
方法是首先选择 View 菜单中的 Utility Windows 项的 Node Finder 选项。在 Filter 框中选择 Pins : all，然后单击 List 按钮。于是在下方的 Nodes Found 窗口中出现设计中的工程的所有端口引脚名。用鼠标将重要的端口节点 a，b，s，co 分别拖到波形编辑窗口，结束后关闭 Nodes Found 窗口，如下图所示：

（4）编辑输入波形（输入激励信号）

单击图示的输入信号 a，使之变成蓝色条，在单击左列的时钟设置键，同理设置其他输入波形，如下图：

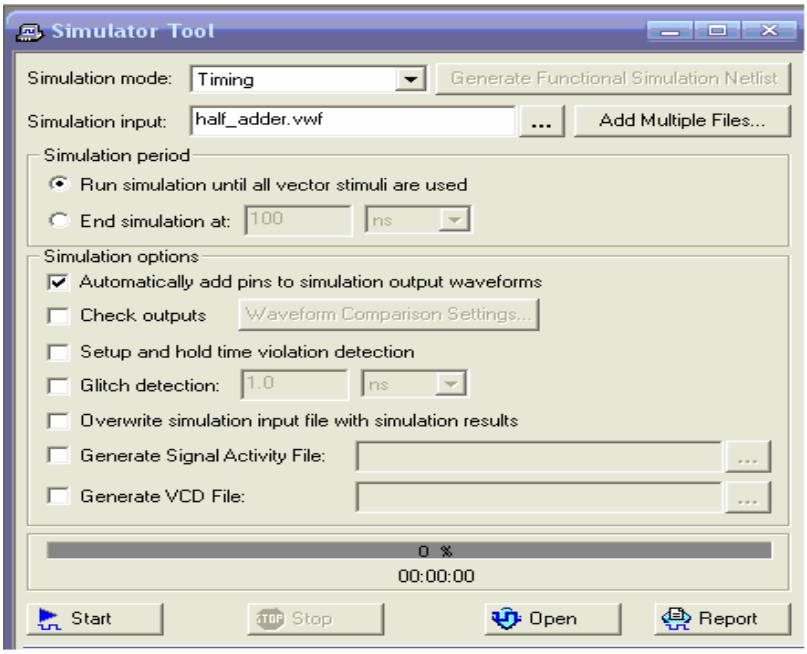


按编译键成功仿真出结果如下：

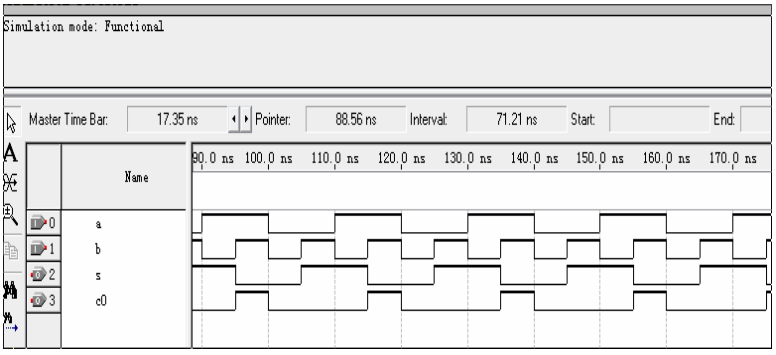


（5）功能仿真

选择 processing→simulator tool，出现以下对话框：

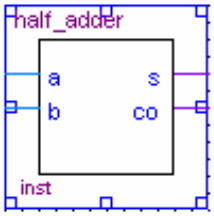


在 simulation mode 选项里，选择 functional，点击 generate functional simulation netist。然后点击 start，进行功能仿真，结果如下：

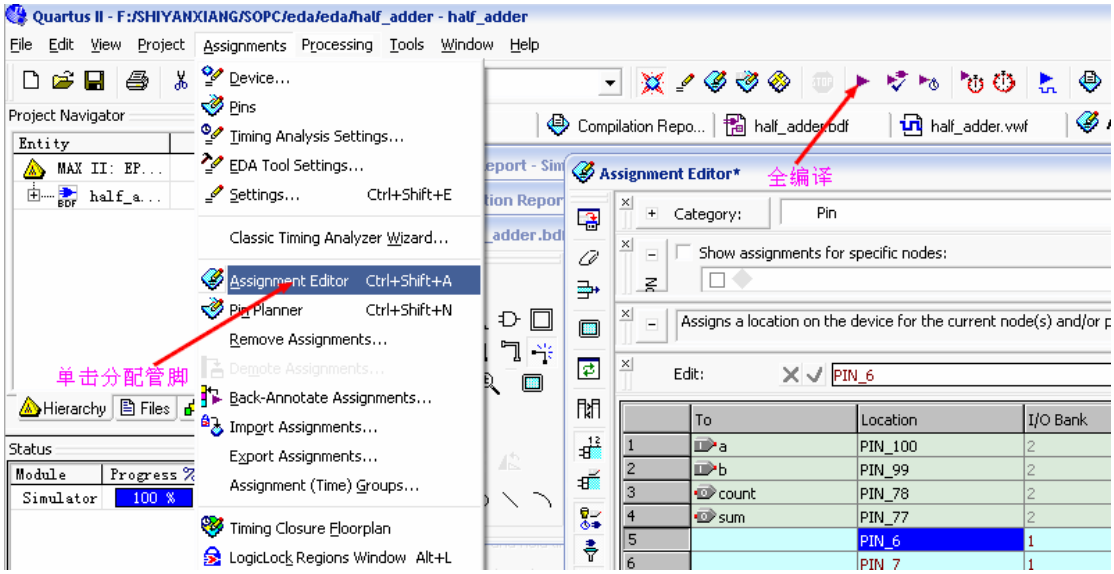


4、生成顶层原理图

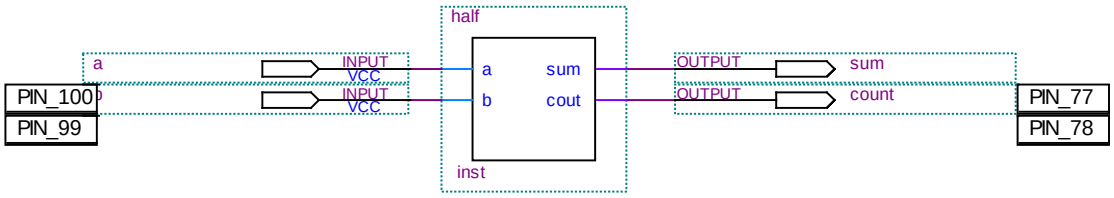
file→creat/update→creat symbol files for current file。建一空白原理图文件，按照上述调出元件的步骤调出生成的半加器图如下：



5、分配管脚如下图所示

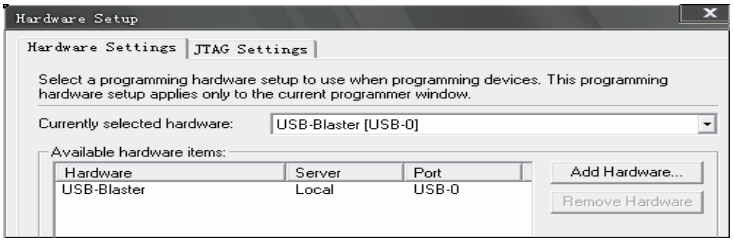


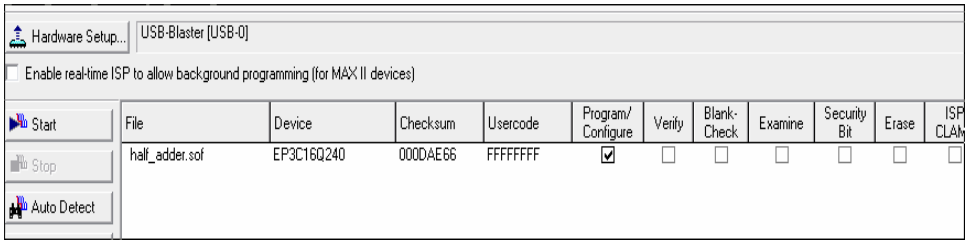
按照上图所示分配管脚，并保存，如下图示：



6、编程下载

管脚分配完毕后，启动全编译，然后下载到 FPGA。首先点击 ”下载” 按钮，检测安装下载电缆(单击 Hardware Setup)弹出对话框如图所示，选择 Byteblaster II 下载电缆，关闭该对话框，然后载入并选中下载文件，点击 Start 开始下载





7、连线

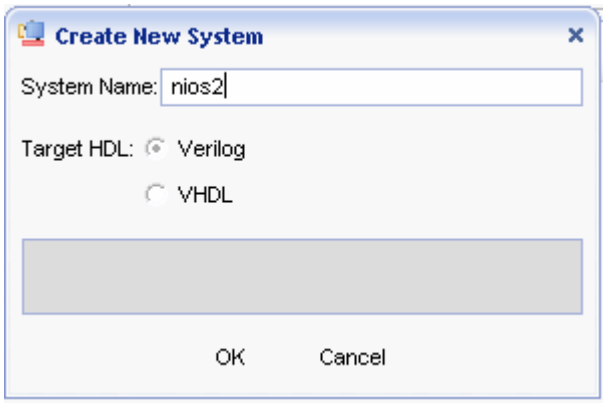
按照分配管脚的顺序使用排线将 PIN-100，PIN-99 分别接入拨码开关 SW1 和 SW2，PIN78，PIN-77 分别接 LED1 和 LED2 。然后手动拨动开关就可以观察半加器两位相加的效果。

二、SOPC Builder / Nios II IDE软件使用方法

1、使用 SOPC Builder 建立 CPU：

Tools->SOPC Builder。

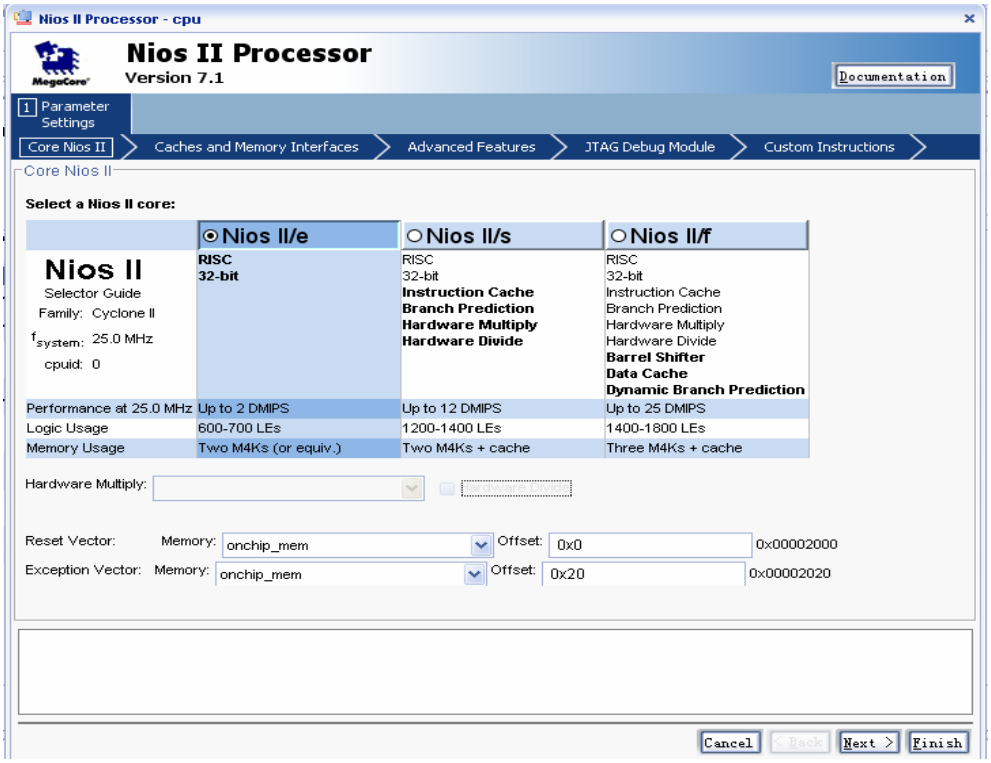
在弹出对话框中给 CPU 重命名如图所示，命名为 NIOS2（例如）；接下来就可以添加所需要的硬件系统模块。



2、添加系统模块组件：

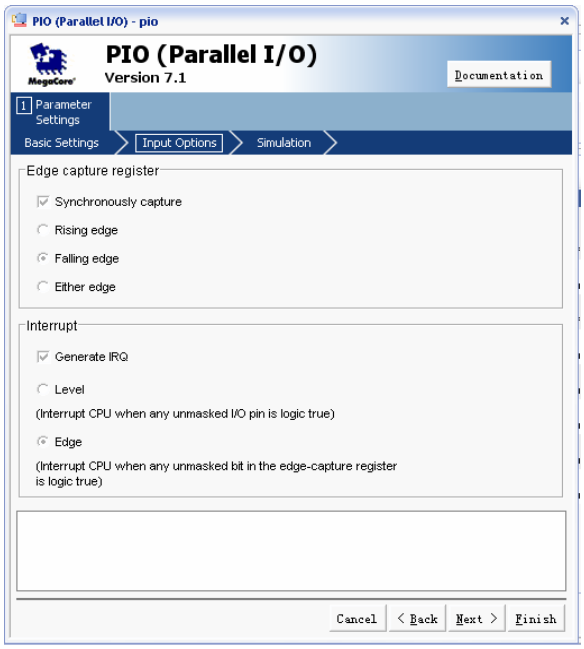
（1）加入 NiosII CPU Core。首先是从左边栏中选择加入 CPU 核 Nios II Processor。选择 SOPC Builder 的组件选择栏中的“Avalon Components”一“Nios II Processoror"，双击鼠标左键。打开添加 NiosII 对话框（如下图），NiosII CPU 核有 4 种结构可以选择，有不同的配置、功能和资源耗用情况，在此点击选择 Nios II/s，再点击 Next，进入窗口 Caches 选择窗，确认设定 instruction cache size 为 4 kbytes，再点击 JTAG Debug Module 栏，最后点击 Finish 按钮完成 NiosII CPU Core 的添加过程。可以看到，niosII 作为一个 CPU Core 组件已经加入 SOPC 系统。更改组件名称的方法是选中“CPU”。

注意，加入组件的更改和取名很重要，许多组件名此系统的工作软件，C 程序中都会出现，而且大小写有关系。



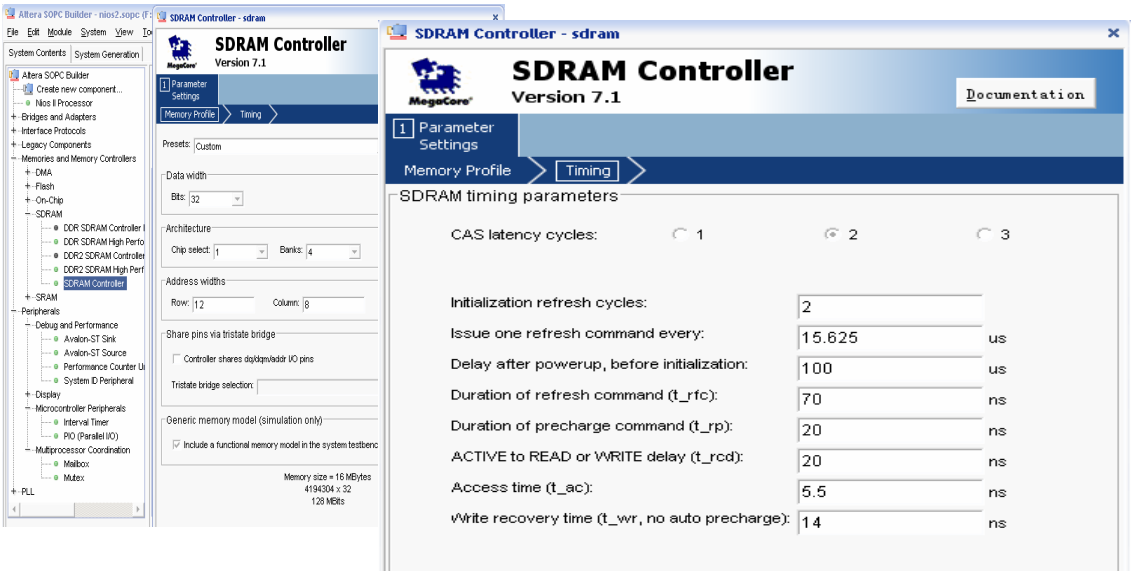
（2）加入 JTAG UART。从左侧组件 Communication 栏中选择的 JTAG UART 加入，接受弹出窗口中的所有默认设置，点击 Finish,完成设置，并改名为 JTAG UART。

（3）加入定时器 Timer。在组件选择栏中选择“peripherals” 一“Microcontroller Peripherals”一"Interval Timer”加入 SOPC 系统的内部定时器。一切都按照默认配置(如下图)，点击 Finish 完成加入。更改组件名称为“sys_clk_timer。此定时器可用于此后在运行的 C 程序中的某些软件函数进行速度。



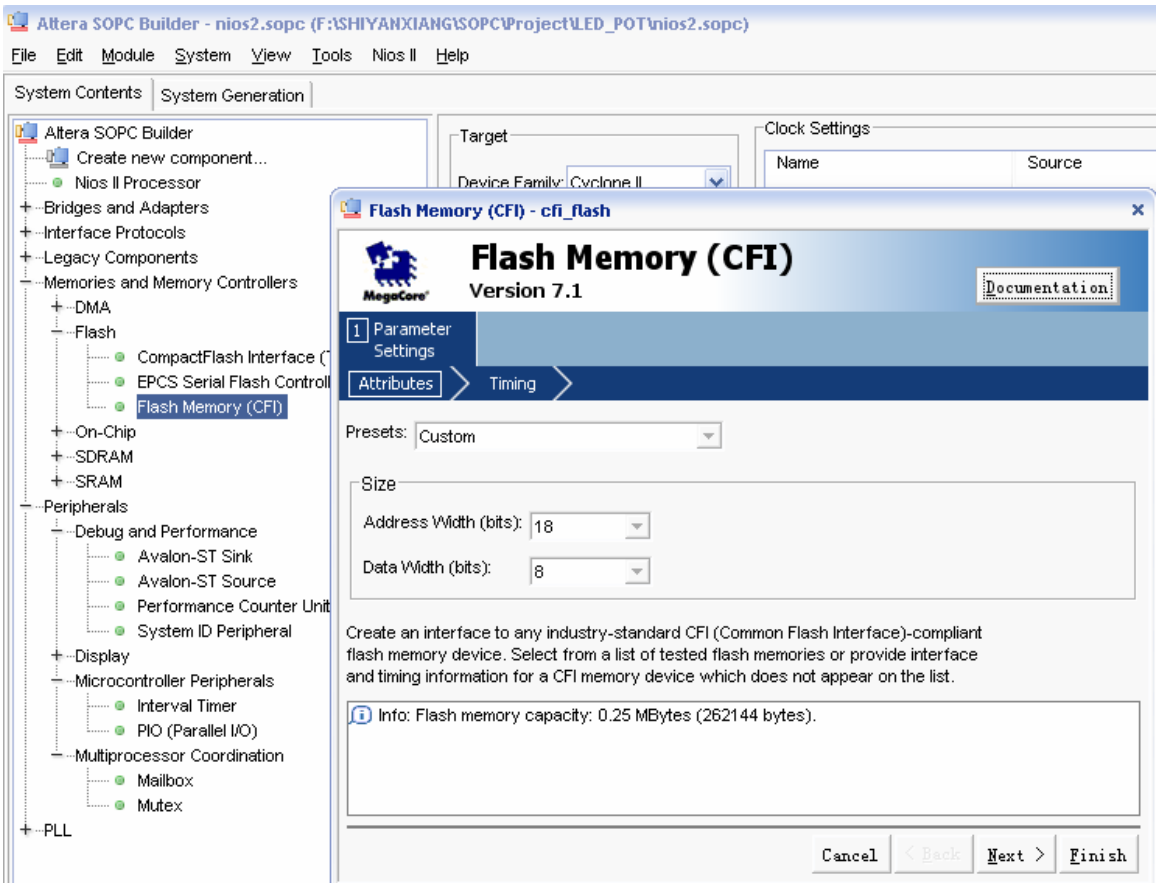
（4）加入键输入 I/O 口。在这里，一般需要加入用于 CPU 的输入输出 I/O 口 PIO。PIO 就是通用 I/O 口。在组件选择栏中选择“peripherals” 一“Microcontroller Peripherals”一“PIO <ParallelI/O)”加入。选择位数，并自定义输入输出。点击“Next”，如下图所示，需要设定输入 I/O 的中断属性。与普通单片机的中断概念相同，有边沿触发和电平触发。点击 Next 后 FINISH。

（5）加入外部 SRAM 组件。在组件选择栏中选择“Memory”----SDRAM，如下图所示。点击 next，设置读写时序。点击“Finish”完成加入。更改组件名称为“sdram”。


SDRAM Controller - sdram
SDRAM Controller
Version 7.1
Documentation
Parameter Settings
Memory Profile
Timing
SDRAM timing parameters
CAS latency cycles: 1 2 3
Initialization refresh cycles: 2
Issue one refresh command every: 15.625 us
Delay after powerup, before initialization: 100 us
Duration of refresh command (t_rfc): 70 ns
Duration of precharge command (t_rp): 20 ns
ACTIVE to READ or WRITE delay (t_rcd): 20 ns
Access time (t_ac): 5.5 ns
Write recovery time (t_wr, no auto precharge): 14 ns

（6）加入 Avalon 三态总线桥。NiosII CPU 与 SRAM，SDRAM，在组件选择栏中选择“Bridg一“Avalon Tri-State Bridge"，加入;"Finish”完成加入。更改组件名称为“Tri-State Bridge“。Flash、自定制组件相接都需要 Avalon 三态总线桥。

（7）加入外部 Flash 。在组件选择栏中选择 “Flash Memory (Common FlashInterface)"如图所示。加入 Flash ROM 组件。在弹出的参数设置窗中，“Attributes"栏中首先选择地址线宽度 Address+width 为 23，和数据线宽度 Data width 为 8，此后 Presets 栏自动选择“Custom"，在“Timing"栏，设置 niosII 对 Flash 的读写时序，此后选择默认设置。点击“Finish“完成加入。更改组件名称为“ext flash”。



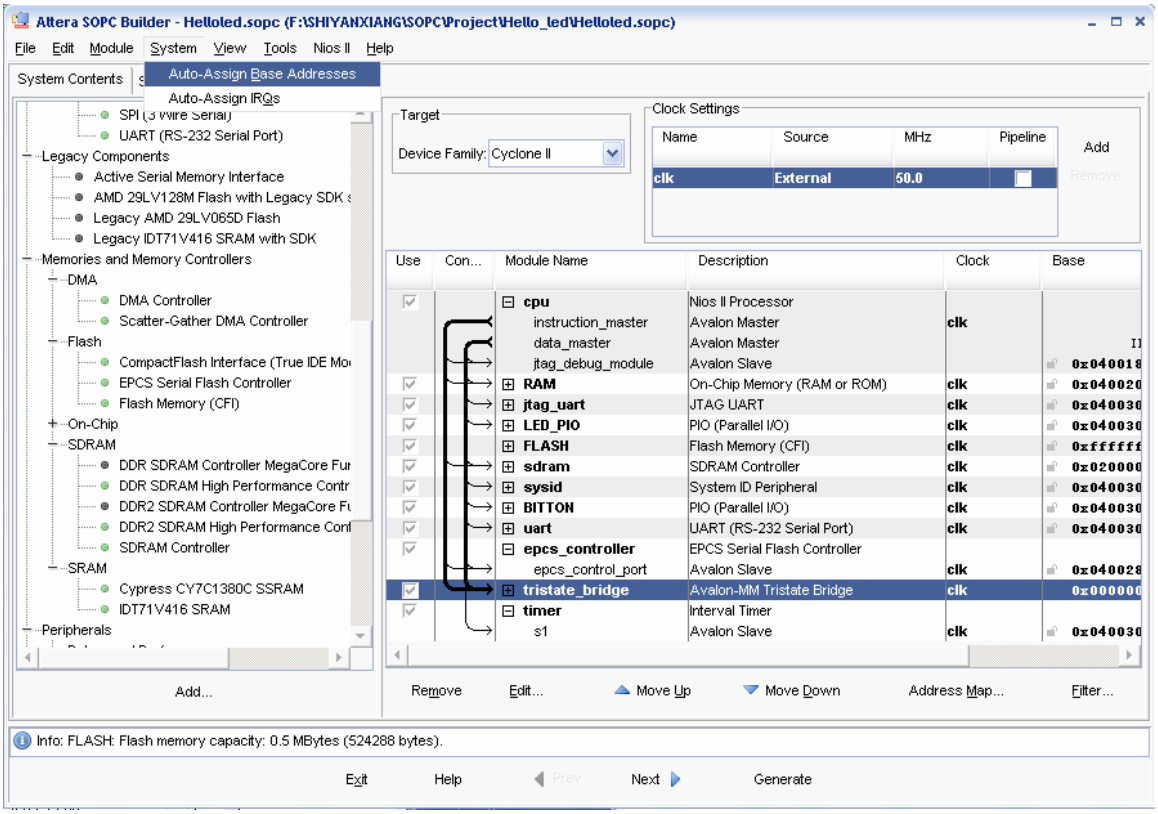
（8）加入系统 ID 组件。在组件选择栏中“peripherals” 一“Debuge and performance”---“System ID Peripheral"，加入此组件，此组件名不可改。

（9）加入 EPCS Serial Flash Controller 组件。EPCS Serial Flash Controller 可用于 NiosII 处理器对 EPCS Flash 存储器的读写访问，可以通过此控制删除 SOF 文件和 POF〕运行的软件一并存于 EPCS 器件中，以便大大简化硬件系统组成结构。在组件选择栏中选择“Memory”一“EPCS Serial Flash Controller"，点击“Finish”完成加入此组件。

（10）NiosII 系统生成前设置与系统生成。

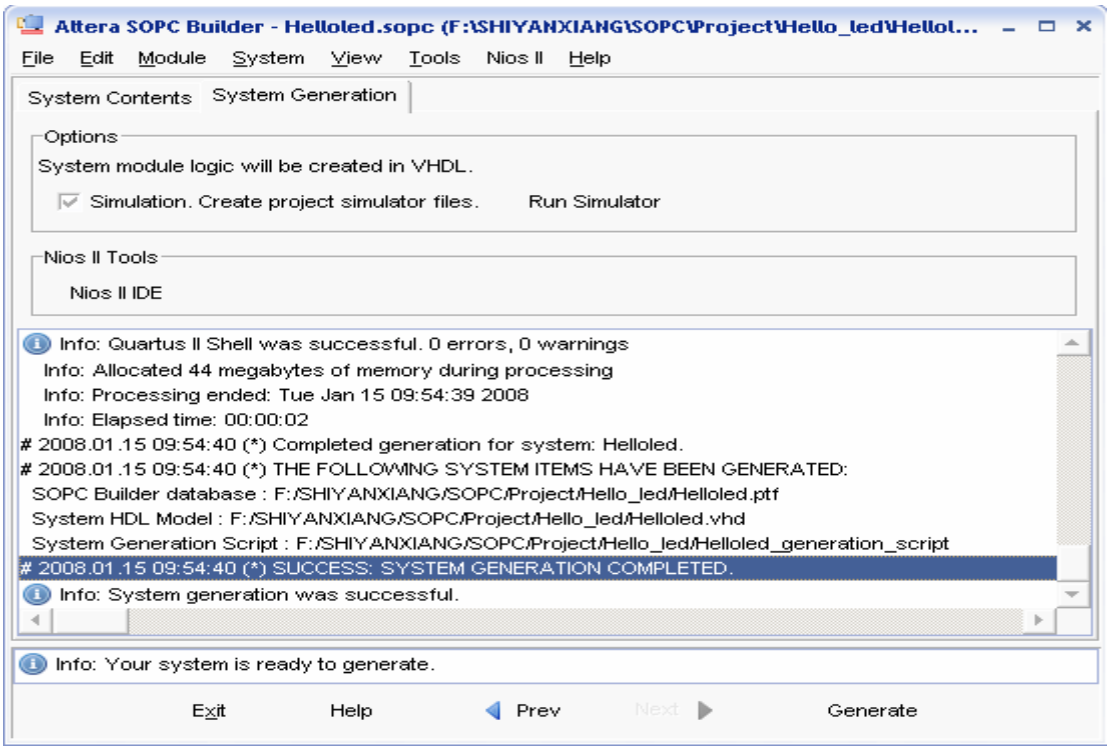
1）地址自动分配设置。为了保证所有组件的地址安排是合法的，如下图所示，选择菜单 System 中的 Auto-Assign Base Addresses 项，对各组件地址进行自动分配，Auto-Assign Irqs 项，对各组件中断优先级进行自动分配，也可以根据需要手动修改。

2）复位地址和程序运行区域置。双击 CPU 点击，进入 NiosII 处理器配置窗。选择 ext flash 作为复位程序区域：ResetAddress；选择 ext sram 作为外部存储单元中的软件运行的存储地址区域：Exception Address。 设置完成后如果有错误可注意到下方的信息栏中已没有早先指出的大量错误提示。



3、系统文件生成。

将最终生成 NiosII 系统的 VHDL 文件，以及对应的硬件仿真文件。点击 Generate 生成过程如下图所示。 注意如图所示信息栏中出现"... SUCCESS: SYSTEM GENERATION COMPLETED"，则可点击 Exit 按钮，退出系统生成窗。



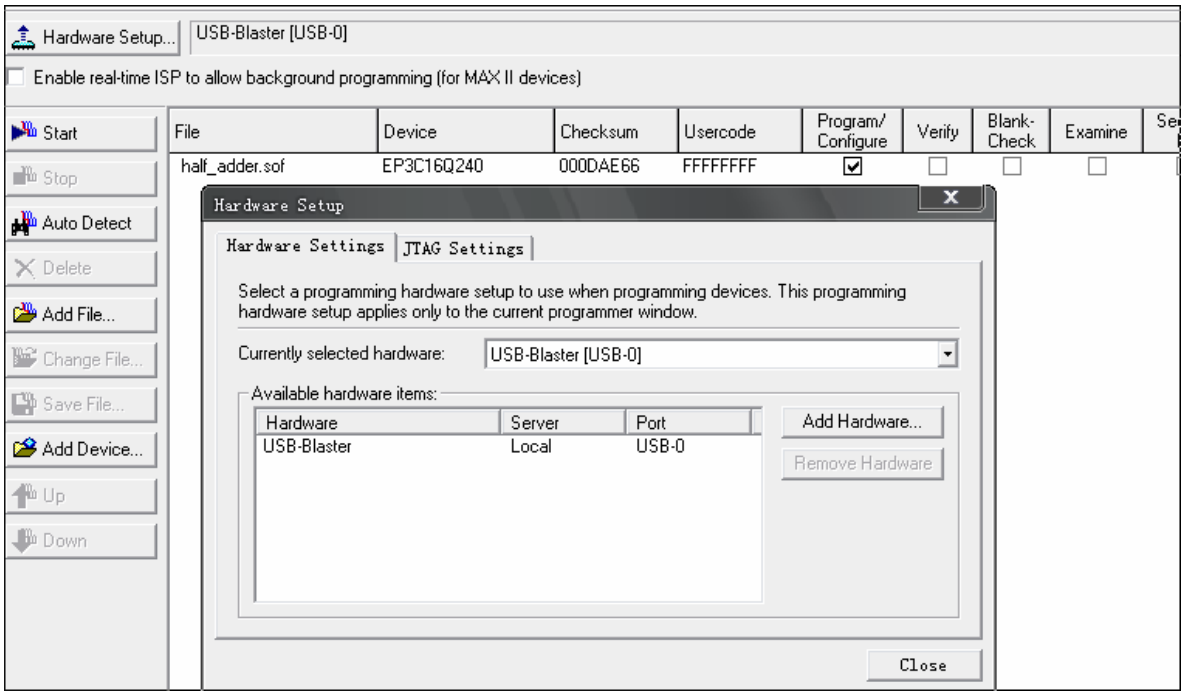
4、NiosII 硬件系统生成。

（1）加入原理图模块。回到原理图模型窗口，在此原理图编辑窗的空白处双击，将弹出元件“Symbol”窗口，点击左侧的元件库栏中的 Project 项，选择刚才生成好的“Nios2”模块，再点击下方的 OK，即可将此元件雕入原理图编辑窗中。将调入的模块于图中的引脚连好，并仔细核对每一引脚都对接正确，包括外部存储器的地址线、数据线、读写控制线、片选线、各类 I/O 口线、复位线。

（2）编译并下载。以上的所有准备工作完成后，就可以进行全程编译了，即进行分析、、适配和输出文件装配，并对结果进行时序分析，给出编译报告。

5、下载。

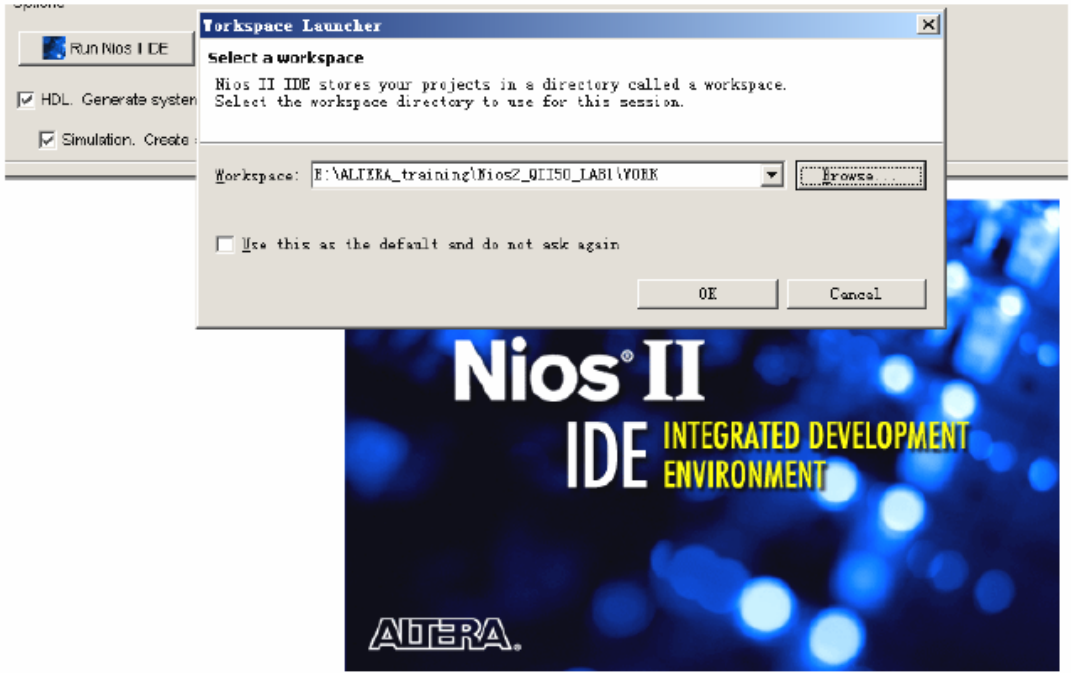
现在可以开始下载配置文件，以便在 FPGA 中建立 NiosII 硬件环境。打开 QuartusII，再打开实验一的工程，选择菜单 Tools-→Programmer，弹出下图所示窗口，首先选择接口模式，对于 Mode 栏确认选择 JTAG 模式；如果是首次安装并使用该软件，要作下载接口模式选择：点击下图左侧的 Hardare Setup 按纽，将弹出图中间所示窗口。如果实验系统的连接是正确的，在图中的窗中的 Hardare 栏应该看到测试到的“ByteBlasterII”或“USB-Blaster”接口名，双击该名再退出该窗。最后加入配置文件*.Sof。点击 Start 按钮，下载该文件。



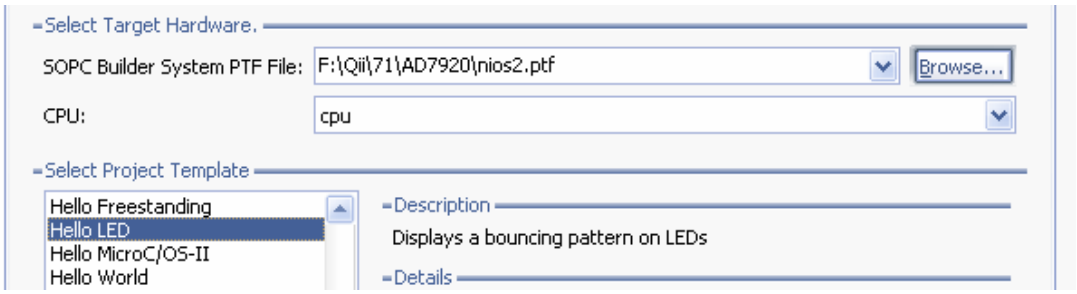
6、进入集成开发环境 Nios II IDE 集成开发环境。

（1）启动软件并建立 c/c++新工程如下图所示。作此选择后将弹出如图所示的窗口，其中有一个软件过程路径选择窗 workspace，在此选择如图所示的路径，是本示例中已预先建立的一个

空文件夹;如果此前已经有了自己的软件实例工程库文件夹，现在还想使用，则必须浏览到对于路径文件夹。点击 OK，再点击此窗右上角的 workbench 按钮，即进入 Nios II IDE 开发环境。

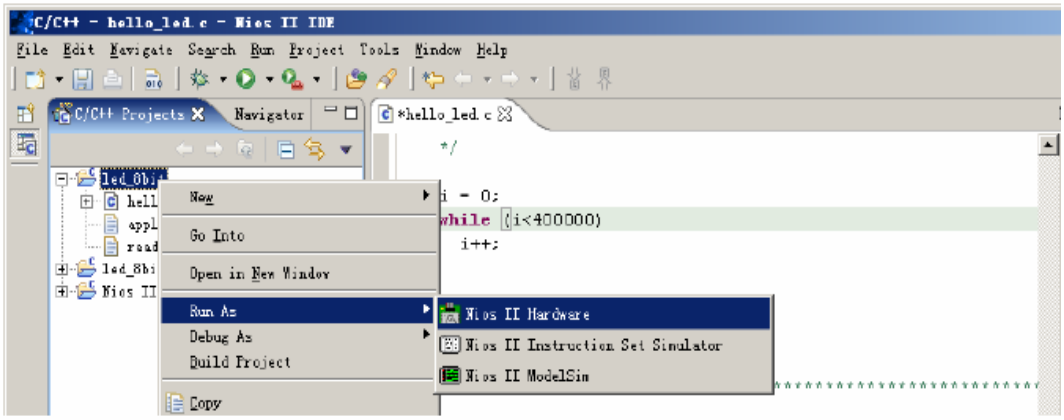


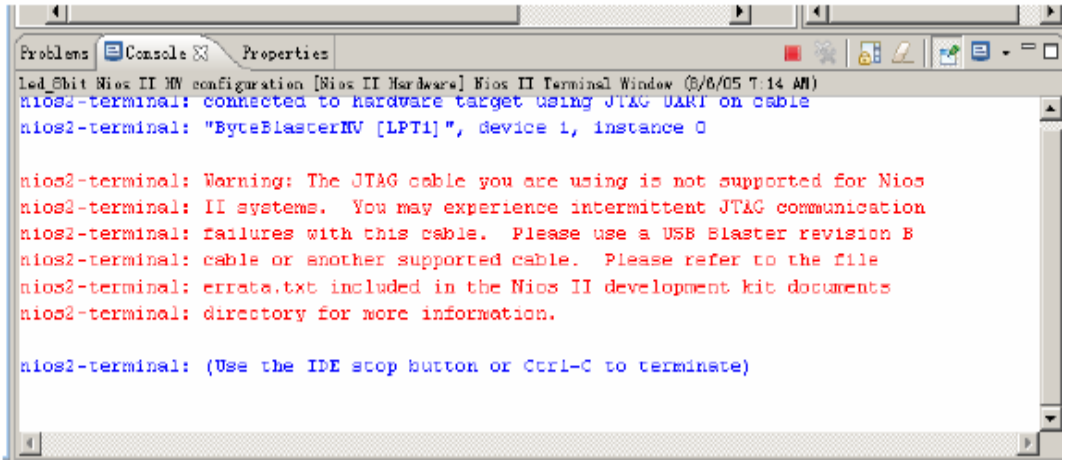
（2）建立 C 软件开发工程。进入的 Nios II IDE 环境窗口的左栏是各工程的工程名和相关的应用文件名，中间是选中的某一文件的内容，及其编辑环境;右栏是对应文件中关键项目名称。为了新建一个开发软件的工程项目，选择菜单 File 的 New->Project.，在弹出的如下图所示的窗口中选择已生成 cpu 的路径。在本例中以 Nios2.ptf 为例。然后选择 Hello led 为例，点击 next-→finish 完成工程建立。



（3）编泽运行 C 程序。

在左边的“C/C++ Projects"页一栏中，右键点击需要运行的工程名:"Hello led "，右键点击该工程名，选择 Run As 后出现另一下拉栏，此栏有 3 个选择项：第一个选项功能是编译并向 FPGA 中的 NiosII CPU 下载和全速运行该工程中 C 程序；第二个选项功能是编译并在虚拟的 NiosII 中运行程序；第三个选项功能是使用第 3 方工具运行；在此选择第一项功能：Run As-> Nios H Hardware，成功运行后结果如下图所示。





第三章 VHDL基本程序设计实验

实验一 组合逻辑半加器的设计

一、 实验目的

2、 通过一个简单的一位半加器的设计，让学生掌握组合逻辑电路的设计方法。

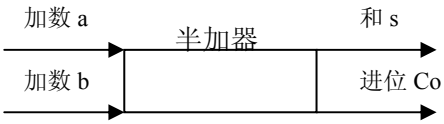
3、 掌握组合逻辑电路的静态测试方法。

4、 初步了解可编程逻辑器件设计的全过程。

5、 熟识 quartus II 软件基本功能的使用。

二、实验原理

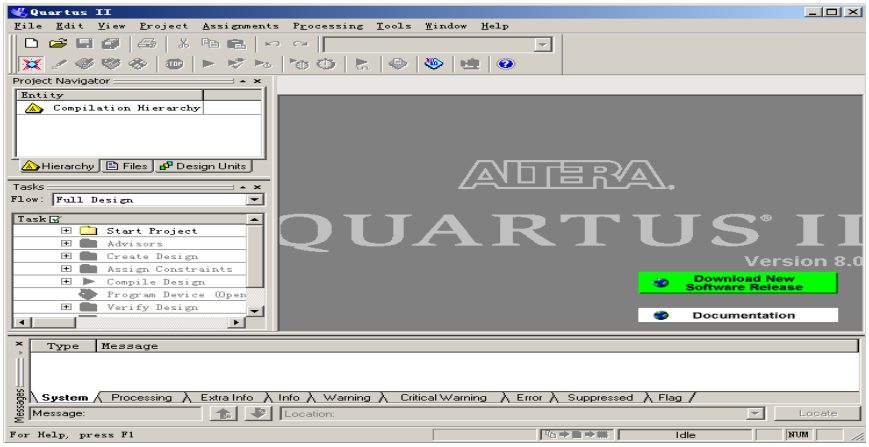
半加器实现两位不带进位加法输出和及进位。一位半加器有两个输入 a、b；两个输出 s，Co。



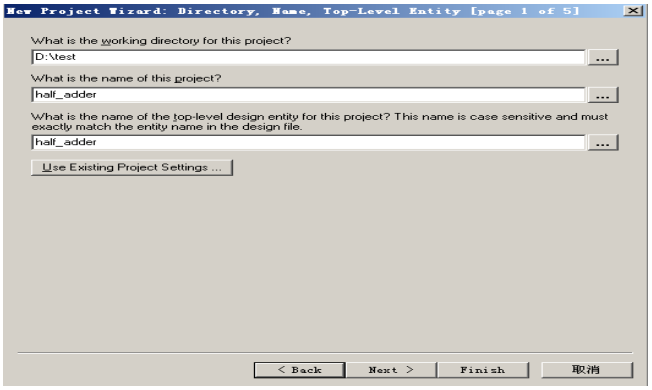
三、实验步骤

1、创建工程

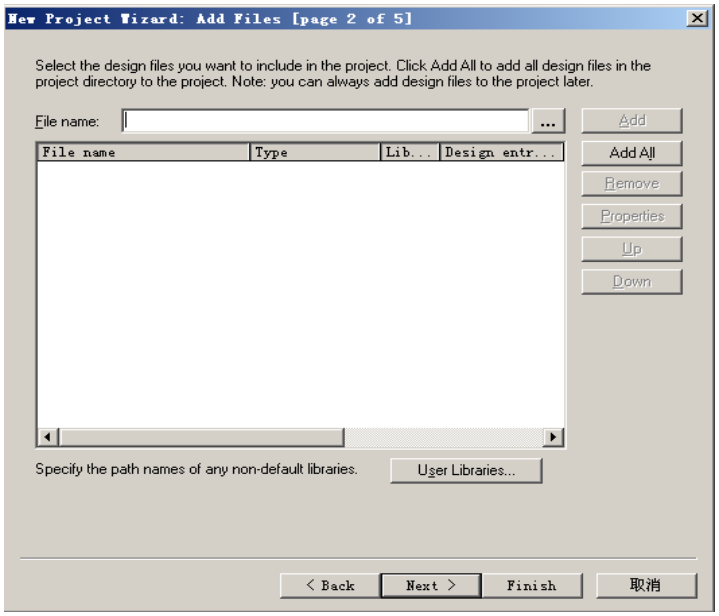
运行 Quartus II 软件，如下图所示：



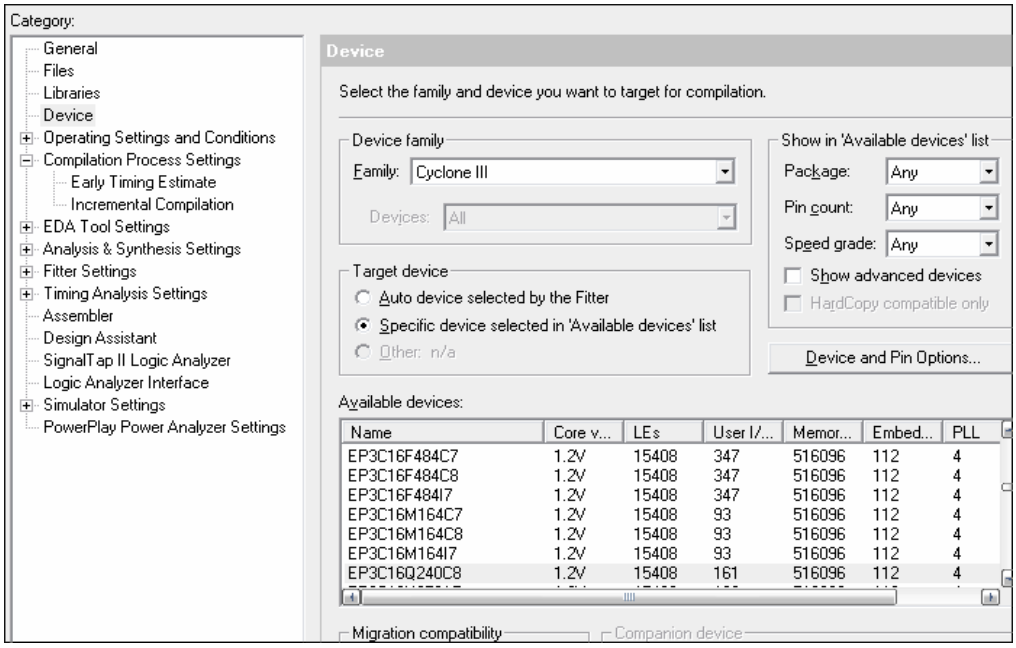
建立工程，File→New Project Wizad，既弹出“工程设置”对话框，如图所示



单击此对话框最上一栏右侧的“...”按钮选择工作目录，在 D 盘中建一个工程文件夹，取名为 test。单击“打开”按钮，在第二行和第三行中写工程名为“half_adder”。按 Next 按钮，出现添加工程文件的对话框，如下图所示：



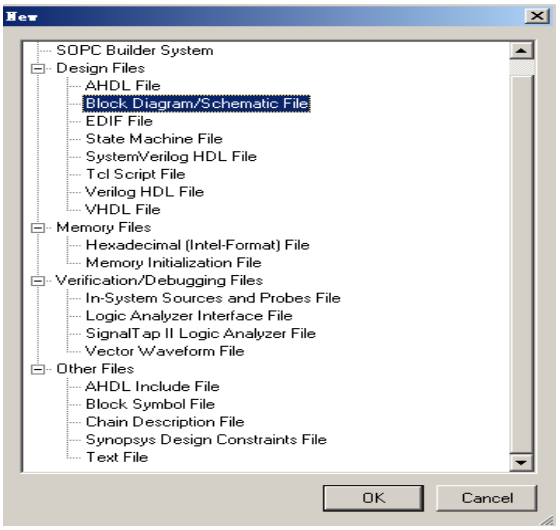
这里无需添加文件，我们跳过此步，直接按 next 进行下一步，如下图所示，选择 FPGA/CPLD 器件的型号，本试验箱选用 Altera 公司的 Cyclone III 系列 FPGA 芯片 EP3C16Q240C8。



在 Family 下拉框中，我们选择 Cyclone III 系列的 FPGA，选择此系列的具体芯片 EP3C16Q240C8。执行 next 出现选择其它 EDA 工具对话框，我们用 Quartus II 的集成环境进行开发，因此这里不作任何改动，连续按 next 进入工程的信息总概对话框，点击 Finish 按钮即建立一个项目。

2、建立顶层原理图

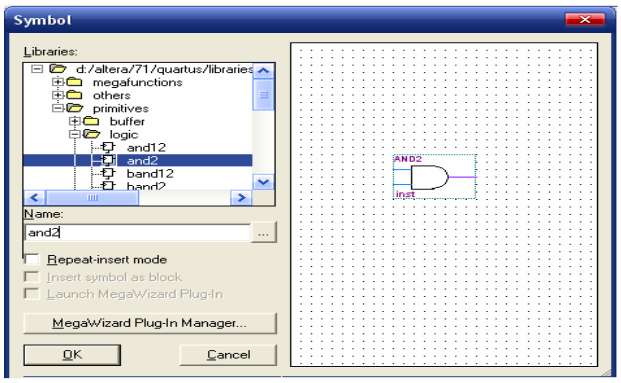
（1）执行 File→New，弹出新建文件对话框，如下图所示：



选择“Block Diagram Schematic File”，按 OK 即建立一个空的顶层图，缺省名为“Block1.bdf”，我们把它另存为（File→Save as），接受默认的文件名，并将“Create new project based on this file”选项选上，以使该文件添加到工程中去。

（2）添加逻辑组件（Symbol）

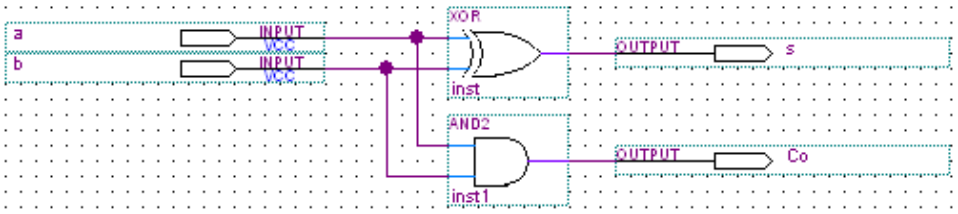
在原理图左边选择 Symbol Tools 或在图的空白处双击，弹出添加组件的对话框，如下图所示。在 libraries 里寻找所需要的逻辑组件，如果知道逻辑组件的名称的话，也可以直接在 Name 一栏敲入名字，右边的预览图即可显示组件的外观，按 OK 后鼠标旁边即拖着一个组件符号，在图纸上点击左键，组件即安放在图纸上。在图中上分别放置异或门（xor），与门(and2)，输入（input），输出（output）。



（3）连线。

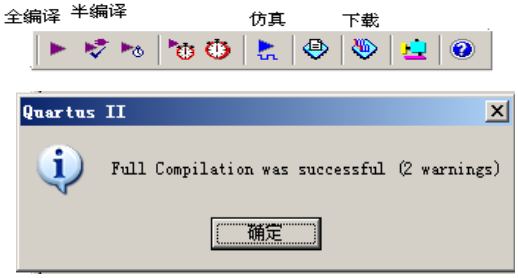
将鼠标移到 symbol 连线端口上，鼠标变成图示形状，按下左键拖动鼠标到另一个 symbol 的连线端。

双击输入输出符号名字“pin_name”，将它们修改为 a，b，s，co，如下图所示：



（4）设置。在建立工程时我们选定的芯片型号，也可以在这一步设定，在菜单 Assignments→Device，如果参数已设置好，这里一般可以不做任何修改。

（5）编译。按主工具栏上的全编译按钮，即开始编译，Message 窗口会显示一些编译信息，最后编译成功弹出提示，如图所示。其中的 warings 可以忽略。

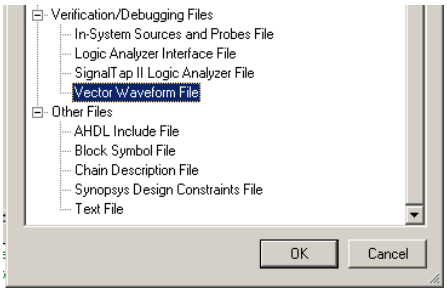


3、仿真

对工程编译通过后，应该对其功能和时序性质进行仿真测试，以了解设计结果是否满足原设计要求。具体步骤如下：

（1）功能仿真

新建波形编辑器。选择菜单 File 中的 New 项，在 New 窗口中选择 Vector Waveform File 项，如下图所示：



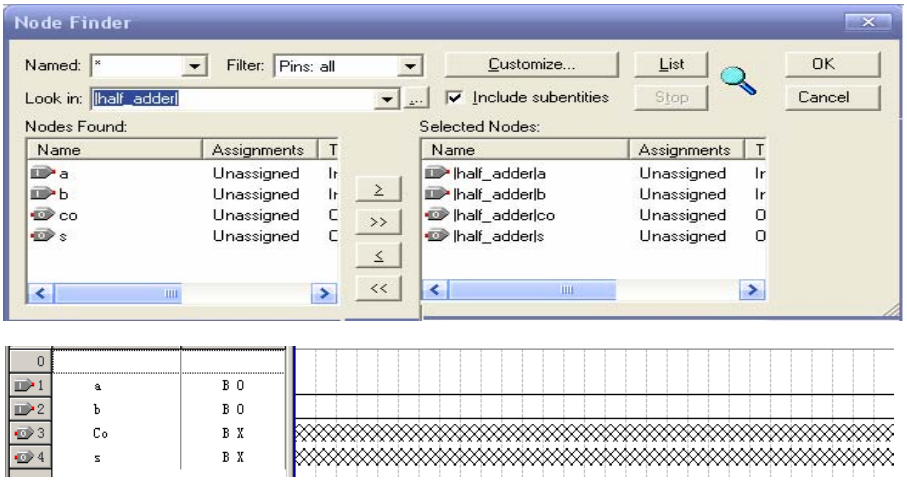
单击 OK 按钮，即出现空白的波形编辑器。设置仿真时间区域。对于时序仿真来说，将仿真时间轴设置在一个合理的时间区域上十分重要。通常设置的时间范围在数十微秒间。

在 Edit 菜单中选择 End Time 项，在弹出的窗口中的 Time 栏处输入 50，单位选择“us”，整个仿真域的时间即设定为 50us，单击 OK 按钮，结束设置。

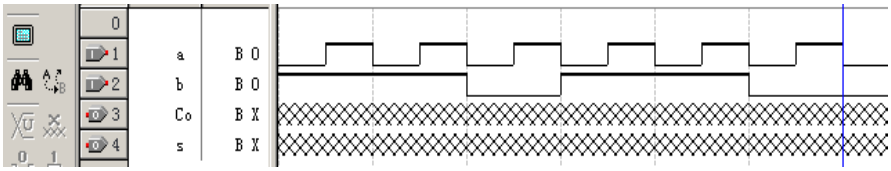
（2）波形文件存盘。选择 File 中的 Save As 项，将以默认名的波形文件存入文件夹中。

（3）将工程的端口信号节点加入到波形编辑器中。首先选择 Edit 菜单中的 Insert 项的 Insert

Node or bus..选项（或在所建立波形文件左边空白处双击鼠标左键再选 Nodes Found）。在 Filter 框中选择 Pins : all，然后单击 List 按钮。于是在下方的 Nodes Found 窗口中出现设计中的工程的所有端口引脚名。用鼠标将重要的端口节点 a，b，s，co 分别加到右边波形编辑窗口，结束后关闭 Nodes Found 窗口，如下图所示：



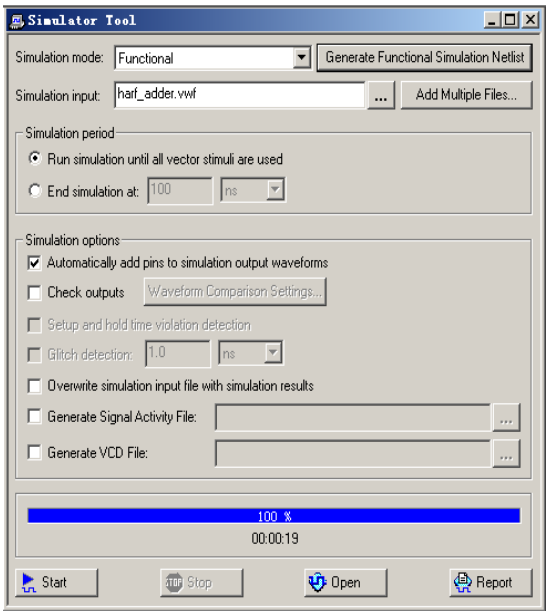
（4）编辑输入波形（输入激励信号）。为输入信号 a、b 添加激励信号，如下图所示。



半编译成功后进行功能仿真

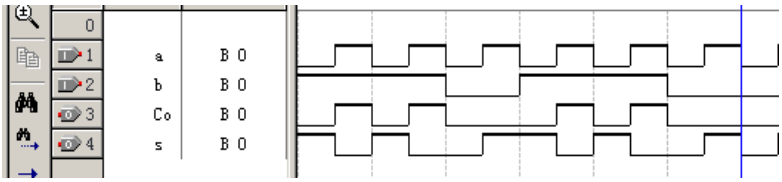
功能仿真：

选择 processing→simulator tool，出现以下对话框：

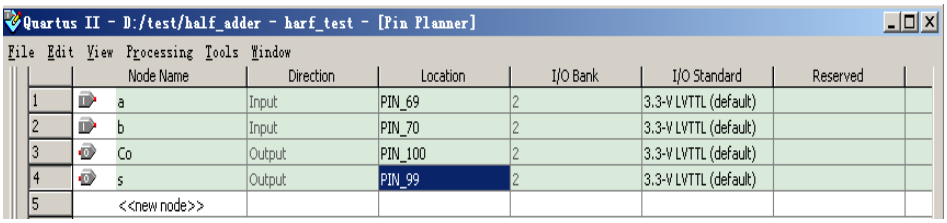


在 simulation mode 选项里，选择 functional，点击 generate functional simulation netist。然

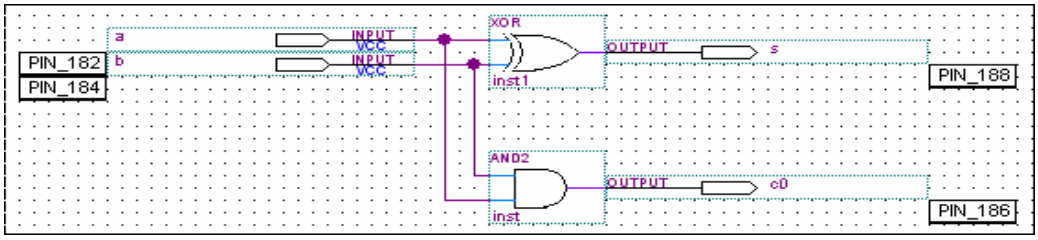
后点击 start，进行功能仿真，成功后点击 Report 查看结果，如下图所示：



4、分配管脚

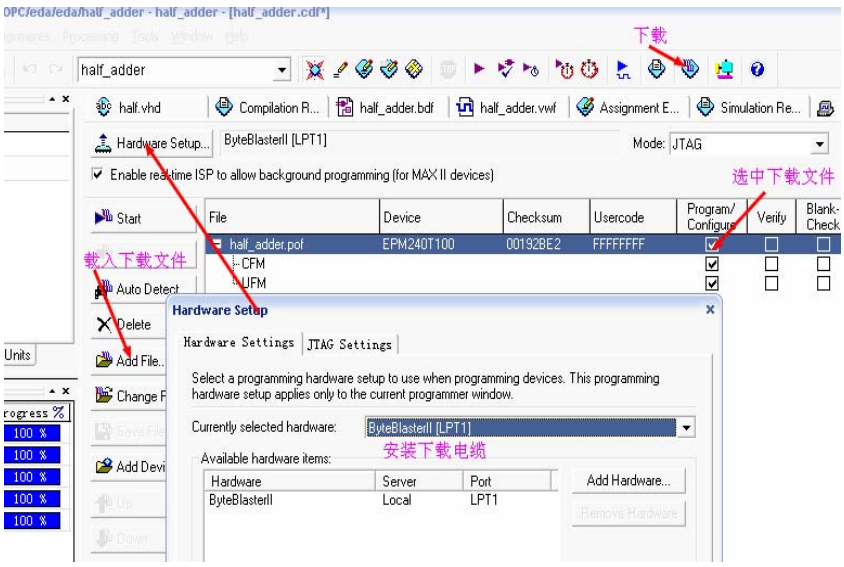


选择 Assignments →Pins Planner 打开如上图所示，并按图中情况分配管脚，按照上图所示分配管脚，并保存。如下图所示：



5、编程下载

管脚分配完毕后，启动全编译，生成 harf_adder.sof 文件，然后下载到 FPGA。首先点击 ”下载” 按钮，检测安装下载电缆(单击 Hardware Setup)弹出对话框如图所示，选择 Byteblaster II 下载电缆，关闭该对话框，（如果已安装就无需再检测下载电缆）载入并选中下载文件，点击 Start 开始下载。



6、连线

按照分配管脚的顺序使用排线将 PIN-182，PIN-184 分别接入拨码开关 SW1 和 SW2：PIN-181，PIN183 分别接 LED1 和 LED2 。然后手动拨动开关就可以观察半加器相加的和和进位的情况。如下表所示

SW1 加数	0	1	1	0
SW2 加数	0	0	1	1
LED1 和	亮	灭	亮	灭
LED2 进位	亮	亮	灭	亮

附：用 VHDL 语言完成半加器的设计

```
library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
        port(a,b: in std_logic;
             sum,cout: out std_logic);
 end half_adder;
 architecture    half1 of half_adder is
 signal c,d:std_logic;
 begin
 c<=a or b;
 d<=a nand b;
 cout<=not d;
 sum<=c and d;
 end half1;
```

实验二　使用VHDL设计组合逻辑全加器

一、实验目的

1、使用 VHDL 语言设计并实现一个一位全加器。

2、初步了解使用 VHDL 进行可编程逻辑器件设计的全过程。

3、熟识 quartus II 软件基本功能的使用。

二、实验原理

加法器一般就是全加器，它实现多位带进位加法。一位全加器有三个输入、两个输出。

真值表：

输入			输出	
Ci-1	Bi	Ai	Si	Ci
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

根据真值表写出逻辑表达式：

$$S_i = A_i \overline{B_i} \overline{C_{i-1}} + \overline{A_i} B_i \overline{C_{i-1}} + \overline{A_i} \overline{B_i} C_{i-1} + A_i B_i C_{i-1}$$

$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1}$$

三、实验步骤

1、 按实验一建立工程命名为 fulladder，并新建 VHDL 文件，File→ new→VHDL File。

2、输入参考程序：

```
library ieee;
use ieee.std_logic_1164.all;
entity fulladder_VHDL is
    port(a,b,cin:in std_logic;
```

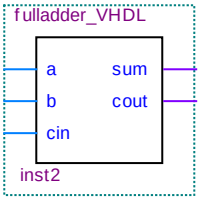
```
                    sum,cout:out std_logic);
            end fulladder_VHDL;
        architecture full1 of fulladder_VHDL is
         begin
              sum<=(a xor b)xor cin;
              cout<=(a and b)or (cin and a)or (cin and b);
         end full1;
```

3、保存 VHDL

注意在程序存盘时其文件名称必须与实体名相同，保存为 fulladder_VHDL.vhdl

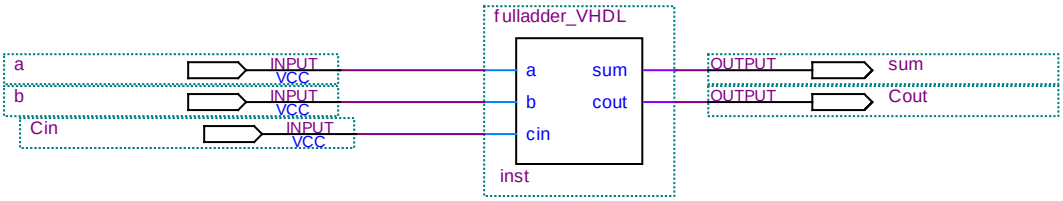
4、生成顶层原理图

选择 File→ Creat/Update→ Creat Symbol Files for Current Files 生成顶层符号文件（如下图所示，然后新建一个 Block Diagramm/Schematic File 文件，在空白处双击在对话框中添加该 Symbol 文件（如图 3.2.4 所示）。



5、添加输入、输出管脚并重命名

在 Block Diagramm/Schematic File 文件的空白处双击，添加输入输出管脚，连接完成再重命名后入下图所示：

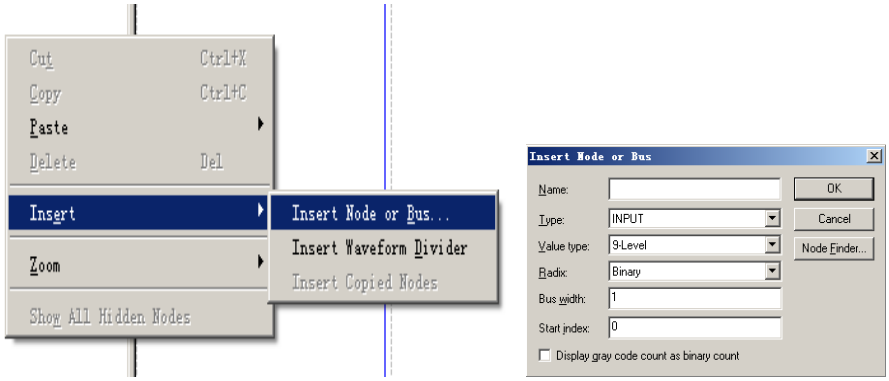


6、保存文件

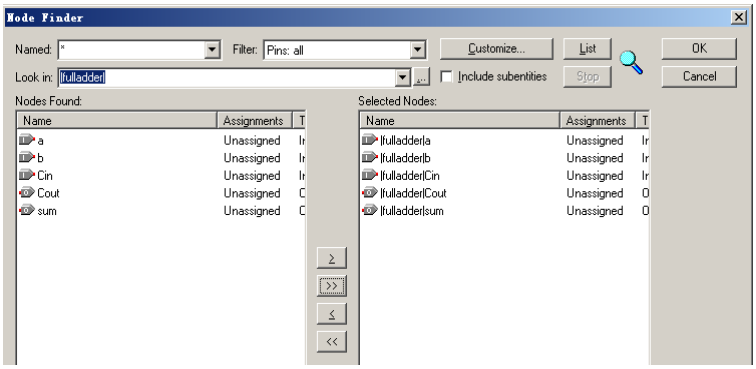
在保存文件时，这里 Block Diagramm/Schematic File 文件是该工程的最顶层文件，保存名应该与工程名相同。VHDL 文件的存储名必须与其实体部分名称相同，当然 VHDL 文件也可以作为顶层文件（这时其实体名称、存储名称、都应该与工程名相同），但是 Block Diagramm/Schematic File 文件 表现出来更直观，所以本实验用 VHDL 文件生成后 Symbl 符号后再使用 Block Diagramm/Schematic File 文件作为最顶层文件。

7、建立仿真文件

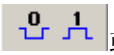
点击主工具栏上的 图标进行半编译，完成后新建一个波形仿真文件 File→New→Verification/Debugging Files→Vector Vaveform File.,然后添加仿真管脚如下图所示。

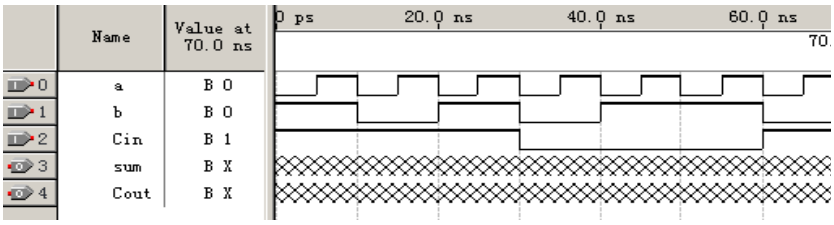


在波形文件左边单击右键如，调出对话框选择 Node Finds，如下图所示。设置如图，点击 List 调出所有管脚，后使用 添加到仿真文件中。点击 OK 后返回到波形文件。



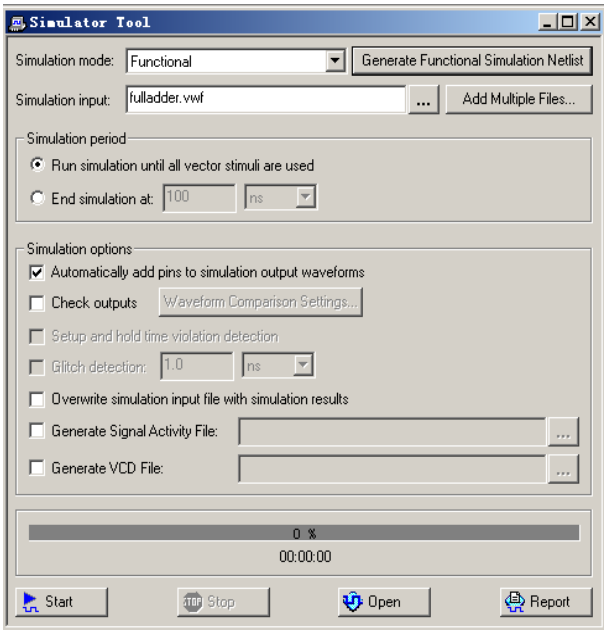
8、添加仿真信号

添加仿真信号如下图所示，首先选中要添加信号的管脚，再利用左边的工具栏的  或 为其添加信号。然后保存仿真文件，命名 fulladder.vwf，然后编译（半编译或全编译）。

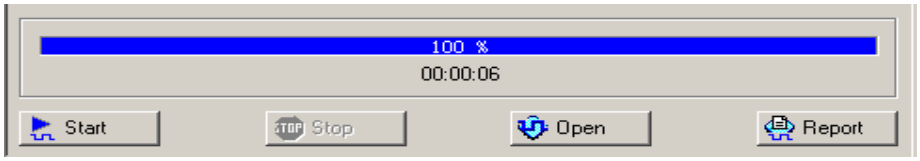


9、波形仿真

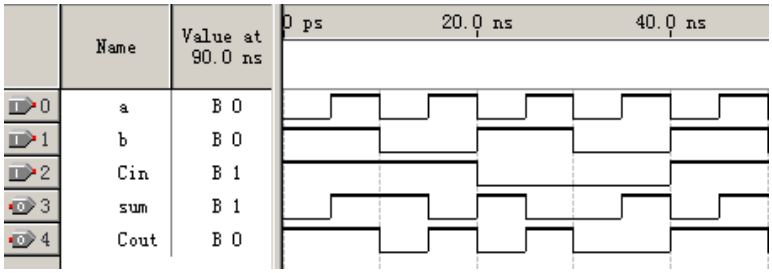
编译完成后，选择 Processing→Simulator Tool 调出仿真对话框如下图所示：



在仿真模式中选择”Functional” 然后点击”Generate Functional Simulation Netlist”生成功能仿真的 Netlist，完成后点击”Start”按钮开始仿真，如下图所示。

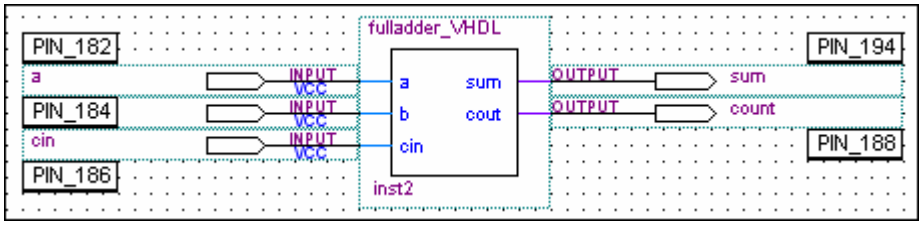


然后点击”Report”来查看仿真结果，如图所示。把仿真结果与真值表相比较观察是否正确。



10、分配管脚

选择 Assignments →Pins Planner 打开并分配管脚，按照图所示分配管脚，分配完毕，保存，然后返回到顶层原理图文件如下图所示。



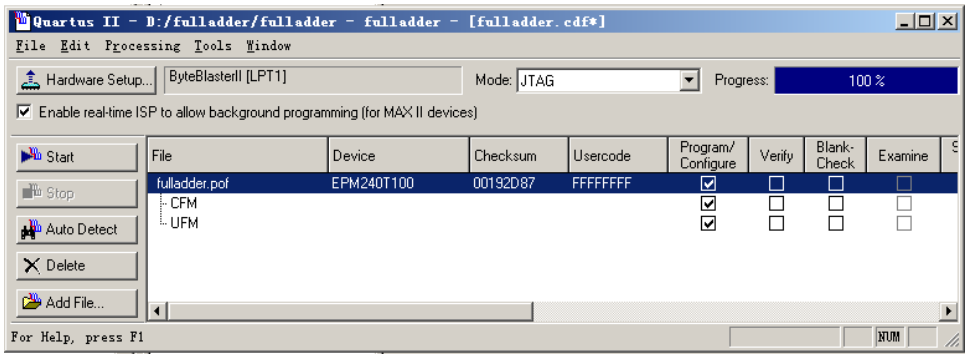
11、保存顶层原理图文件后进行全编译 。

12、下载。

全编译完成后会生成一个 fulladder.pof 文件，将此文件下载到 FPGA 中。

首先点击主工具栏的 图标或选择 Tools→Programmer 来打开下载对话框。如果在左上脚显示有"No Hardware" 则需要安装下载电缆，点击"Hardware Setup"来安装。安装后返回下载对话框后，在 Programmmer/configure 下"打勾"选中下载文件，然后点击"Start"按钮开始下载。

13、下载完成



14、连线并观察记录实验结果

按照管脚分配顺序，把 Pin69、Pin70、Pin71 分别连接拨码开关 SW1、SW2、SW3；而 Pin99、Pin100 分别连接 LED1、LED2。完成后得出以下现象。

开关量输入			指示输出	
SW1	SW2	SW3	LED1	LED2
0	0	0	亮 0	亮 0
1	0	0	灭 1	亮 0
0	1	0	灭 1	亮 0
1	1	0	亮 0	灭 1
0	0	1	灭 1	亮 0
1	0	1	亮 0	灭 1
0	1	1	亮 0	灭 1
1	1	1	灭 1	灭 1

实验三　带进位输入的 8 位加法器

一、实验目的

设计并实现一个 8 位加法器。

二、实验原理

在下图中，“进位入”Ci-1 是指低位的进位输出，“进位出”Ci 即是本位的进位输出。

原理图如下：

三、实验步骤

在本实验中直接调用一位全加器模块，程序如下（程序中设变量 n，设计的是 n 位加法器，改变 n 的值，从而得到多位加法器，本实验中 n 赋值为 8）：

```
library ieee;
use ieee.std_logic_1164.all;
entity fulladder_VHDL is
        port(a,b,cin:in std_logic;
             sum,cout:out std_logic
            );
end fulladder_VHDL;
    architecture full1 of fulladder_VHDL is
begin
sum<=(a xor b)xor cin;
cout<=(a and b)or (cin and a)or (cin and b);
end full1;
```

1、建立工程命名为 fulladder8,然后新建一个 VHDL 文件输入全加器代码（同实验二）：保存文件，命名为 fulladder_VHDL.vhdl（与实体部分名相同）。

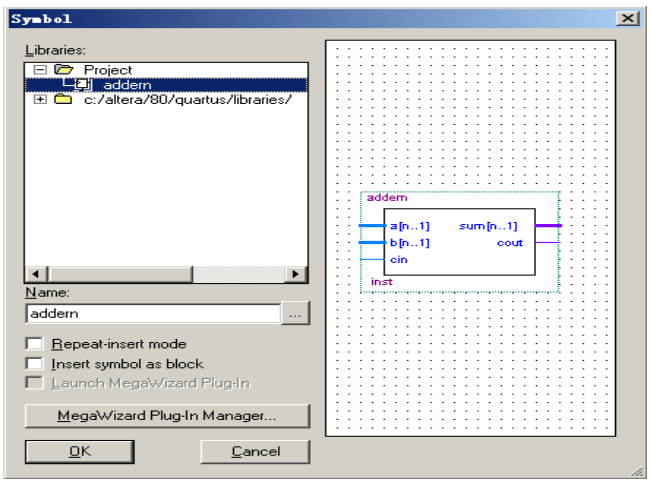
2、再新建一个 VHDL 文件，输入以下代码，来调用全加器程序，保存为 addern.vhdl（与实体部分名相同）。

```
library ieee;
use ieee.std_logic_1164.all;
entity addern is
```

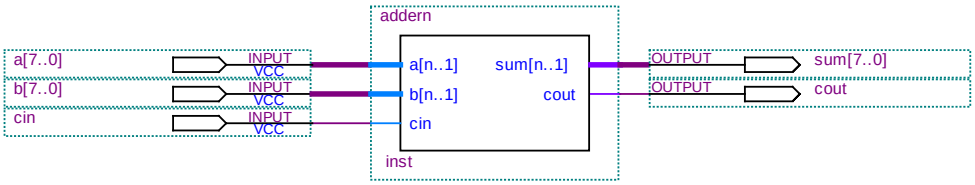
```
        generic(n:integer:=8);          ----本实验赋值 8 位加法器
        port(a,b:in std_logic_vector (n downto 1);
             cin:in std_logic;
             sum:out std_logic_vector(n downto 1);
             cout:out std_logic);
end   addern;
architecture aza2 of addern is
        component fulladder_VHDL     -----声明要调用的 1 位全加器
          port(a,b,cin:in std_logic;
               sum,cout:out std_logic);
        end component fulladder_VHDL;
  signal carry:std_logic_vector(0 to n);
  begin
        carry(0)<=cin;
        cout<=carry(n);
        gen:for i in 1 to n generate
        add:fulladder_VHDL ----开始调用全加器
         port map(a=>a(i),b=>b(i),cin=>carry(i-1),sum=>sum(i),cout=>carry(i));
         end generate gen;
  end aza2;
```

3、把 addern.vhdl 文件生成原理图文件

打开 addern.vhdl 文件，选择 File→ Creat/Update→ Creat Symbol Files for Current Files 生成顶层符号文件，然后新建一个 Block Diagramm/Schematic File 原理图文件，在空白处双击在对话框中添加该 Symbol 文件，如下图所示，点击 OK 后返回。

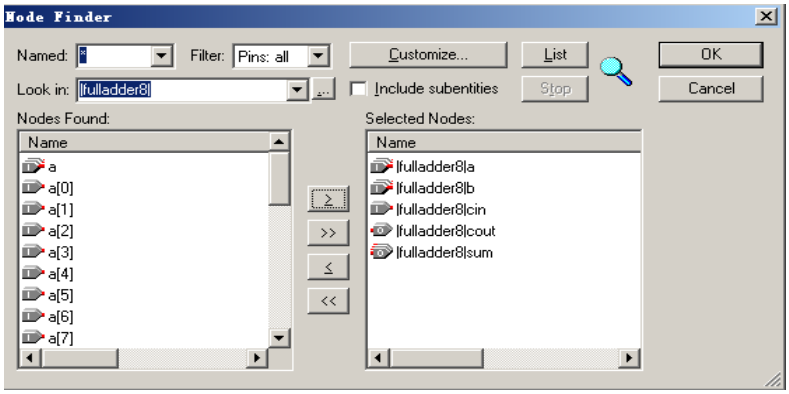


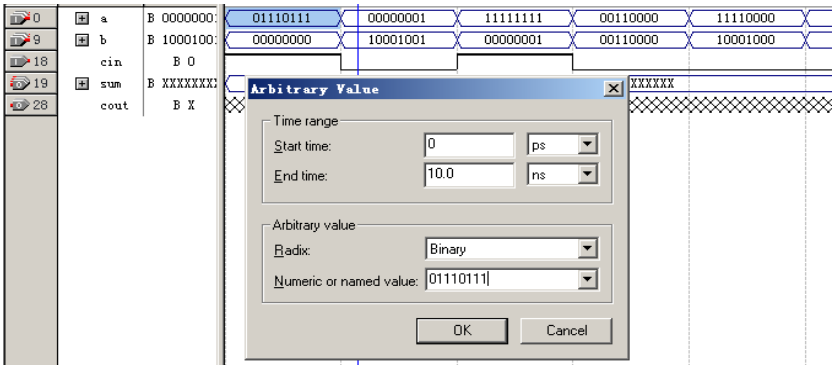
4、 同步骤 3，继续添加 input、output（输入输出）管脚，并重命名后如下图所示。保存原理图文件，命名为 fulladder8.bdf（与工程名相同）。



5、仿真文件

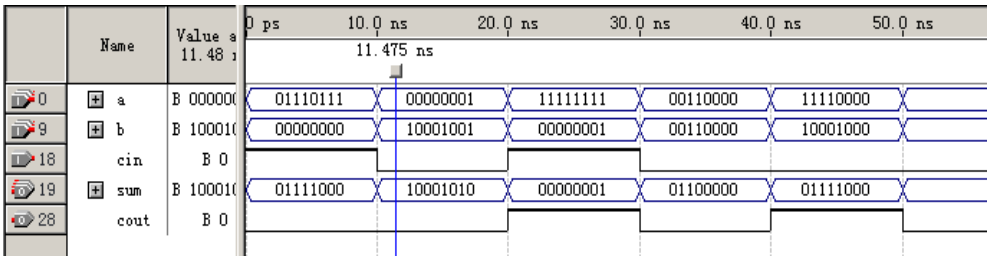
点击主工具栏上的 图标进行半编译，完成后新建一个波形仿真文件 File→New→Verification/Debugging Files→Vector Vaveform File.,然后在左边空白处双击左键添加仿真管脚，完成后添加激励信号的波形，首先使用鼠标选中信号的一个区间（变蓝），后左键双击，输入二进制的数值，现在以信号 a 的一个区间赋值“11110000“为例，如图依次添加各个信号（可以根据老师的实际要求添加不同的信号），并保存，如下图所示。





6、功能仿真

选择 Processing→Simulator Tool 在弹出对话框中在仿真模式中选择"Functional" 然后点击"Generate Functional Simulation Netlist"生成功能仿真的 Netlist，完成后点击"Start"按钮开始仿真，然后点击"Report"来查看仿真结果如下图所示。



7、分配管脚后进行全编译，再下载到 FPGA，并连线。

（管脚分配选择 Assignments →Pins Planner 在 Location 中选择要分配的管脚参考实验一和实验二）全加器的 17 个输入所对应的管脚同 17 位拨码开关相连，17 个输入管脚是 a0～a7、b0~b7 和 cin，a0～a7、b0~b7 代表两个 8 位二进制数，cin 代表进位位；9 个输出所对应的管脚同 9 位发光二极管相连，9 个输出管脚是 sum0~sum7 和 cout，sum0~sum7 代表相加结果，cout 代表进位。

管脚分配：

Pin182—a0 pin184—a1 pin186—a2 pin188—a3 pin194—a4 pin196—a5

Pin198—a6 pin200—a7

Pin181—b0 pin183—b1 pin185—b2 pin187—b3 pin189—b4 pin195—b5

Pin199—b6 pin201—b7

Pin202—sum0 Pin207—sum1 Pin216—sum2 Pin218—sum3 Pin220—sum4

Pin222—sum5 Pin224—sum6 Pin230—sum7

Pin214—cout pin203—cin

8、实验记录

输入			输出 LED	
Cin	Bi(7..0)	Ai(7..0)	Sum（7..0）	Cout
1	00000000	01110111		
0	10001001	00000001		
1	00000001	11111111		
0	00110000	00110000		
0	10001000	11110000		

实验四　全减器

一、实验目的

设计并实现一个一位减法器

二、实验原理

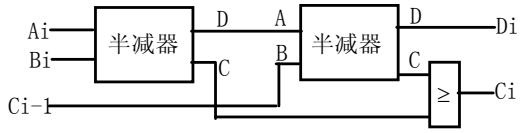
半减器不考虑低位向本位的借位。一位半减器由两个输入、两个输出。

表 1　半减器真值表

输入		输出	
Bi	Ai	Di	Ci
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

由真值表可得到函数表达式：$Di = \overline{Ai}Bi + Ai\overline{Bi}$　　$Ci = \overline{Ai}Bi$

在下图中，“进位入”Ci-1 是指低位的进位输出，“进位出”Ci 即是本位的进位输出。



原理图如下：

真值表：

输入			输出	
Ci-1	Bi	Ai	Si	Ci
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

根据真值表写出逻辑表达式：

$$D_i = \overline{A_i}B_iC_{i-1} + A_i\overline{B_iC_{i-1}} + A_iB_iC_{i-1}$$

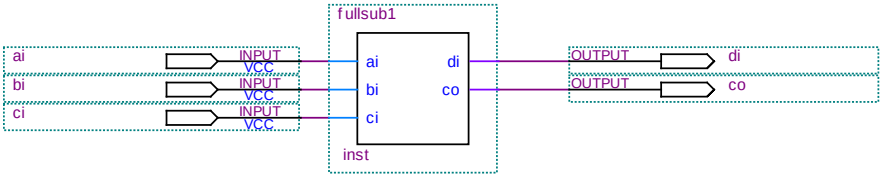
三、实验步骤

1、建立工程 fullsub,新建 VHDL 文件输入以下代码保存为 fullsub1.vhdl。打开 addern.vhdl 文件，选择 File→ Creat/Update→ Creat Symbol Files for Current Files 生成顶层符号文件。（参考实验一、二）

```
----------------------半减器程序
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity halfsub is
port(a,b:in std_logic;
      d,c:out std_logic);
end halfsub;
architecture half1 of halfsub is
begin
      d<=((a and (not b))or ((not a)and b));
      c<=(not a)and b;
end half1;
----------------------------------全减器程序
library ieee;
use ieee.std_logic_1164.all;
entity fullsub1 is
        port(ai,bi,ci:in std_logic;
              di,co:out std_logic);
end fullsub1;
architecture full1 of fullsub1 is
        component halfsub-----------声明半减器调用
             port(a,b:in std_logic;
```

```
                d,c:out std_logic);
end component;
signal halfsub1_d,halfsub1_c,halfsub2_c:std_logic;
begin------------------------------------------------------------调用半减器
    halfsub1:halfsub port map (ai,bi,halfsub1_d,halfsub1_c);
    halfsub2:halfsub port map (halfsub1_d,ci,di,halfsub2_c);
    co<=(halfsub1_c or ci);
end full1;
```

2、新建一个 Block Diagramm/Schematic File 原理图文件，，在空白处双击添加生成的顶层原理图，并连接 input、output（输入输出管脚）重命名后如图所示，保存文件 fullsub.bdf。

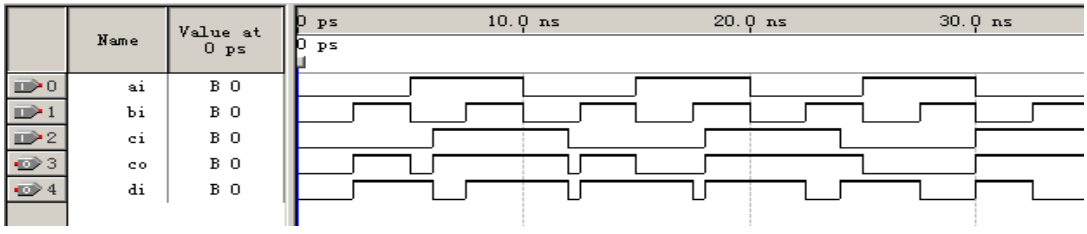


3、 建立仿真文件

点击主工具栏上的 图标进行半编译，完成后新建一个波形仿真文件 File→New→Verification/Debugging Files→Vector Vaveform File.,然后在左边空白处双击左键添加仿真管脚。完成后添加激励信号的波形，首先使用鼠标选中一个信号（变蓝），利用左边工具添加仿真信号，根据需要依次添加各个信号（可以根据老师的实际要求添加不同的信号），并保存。

4、功能仿真

选择 Processing→Simulator Tool 在弹出对话框中在仿真模式中选择”Functional” 然后点击”Generate Functional Simulation Netlist”生成功能仿真的 Netlist，完成后点击”Start”按钮开始仿真，完成后点击”Report”来查看仿真结果，如图所示。



5、分配管脚，下载、连线

选择 Assignments →Pins Planner 在 Location 中选择要分配的管脚，分配完毕后，点击主工具栏的 图标进行全编译，完成后点击 下载到目标器件。连线时 ai 、bi、ci 分配的管脚连接

拨码开关，Co、Di 所对应管脚连接 LED 指示灯。

6、实验记录

根据仿真结果和实验 led 发光二极管的亮灭完成下表，并分析其运算结果的正确性。

输入			输出		实验结果	
C_{i-1}	Bi	Ai	Di	Ci	Di LED	Ci LED
0	0	0	0	0		
0	0	1	1	0		
0	1	0	1	0		
0	1	1	0	1		
1	0	0	1	0		
1	0	1	0	1		
1	1	0	0	1		
1	1	1	1	1		

管脚分配：

Pin182—ai pin184—bi pin186—ca pin188—co pin194—d

实验五　四位向量加法/减法器

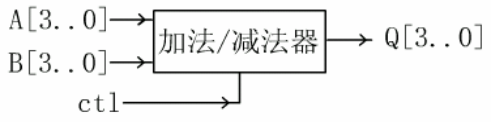
一、实验目的

设计并实现一个四位向量加法/减法器

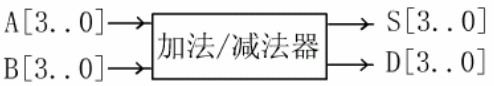
二、实验原理

在前面的几个实验中，我们已经设计了单独的加法器和减法器，那如何把它们通过一个设计就能实现呢？在这里，我们通过一个 4 位的向量加法/减法器来说明，对于多位的运算器，其设计方法相同，不同的就是位宽不同。

对于加法/减法器可以有设计成为一种是带控制信号的运算器（即通过控制信号的高低电平来控制这个运算是加法器还是减法器），另一种是运算器对输入的两个操作数同时进行加法和减法操作。两种运算器的框图如图所示。



（注：当 ctl=1 时，运算器为加法器，为 0 时为减法器）



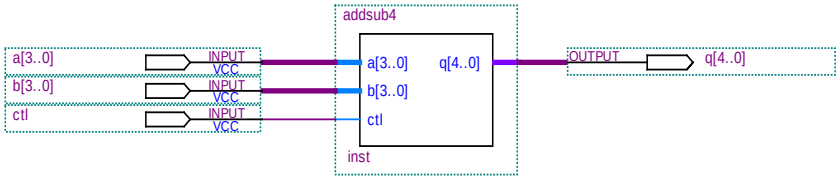
三、实验步骤

1、建立工程 addsub，新建 VHDL 文件输入以下代码保存为 addsub4.vhdl。打开 addsub4.vhdl 文件，选择 File→ Creat/Update→ Creat Symbol Files for Current Files 生成顶层符号文件。（参考实验一、二）

```
library ieee;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity addsub4 is
      port(a,b:in std_logic_vector(3 downto 0);
            ctl:in std_logic;
            q:out std_logic_vector(4 downto 0));
end addsub4;
architecture   add_sub1 of addsub4 is
  begin
```

```
        process(a,b,ctl) ---
          begin
            if ctl='1' then
                q<=('0'&a)+b;
            elsif ctl='0' then
                q<=('0'&a)-b;
            end if;
        end process;
end add_sub1;
```

2、然后新建一个 Block Diagramm/Schematic File 原理图文件，，在空白处双击添加生成的顶层原理图，并连接 input、output（输入输出管脚）重命名后如图所示。保存文件 fullsub.bdf。

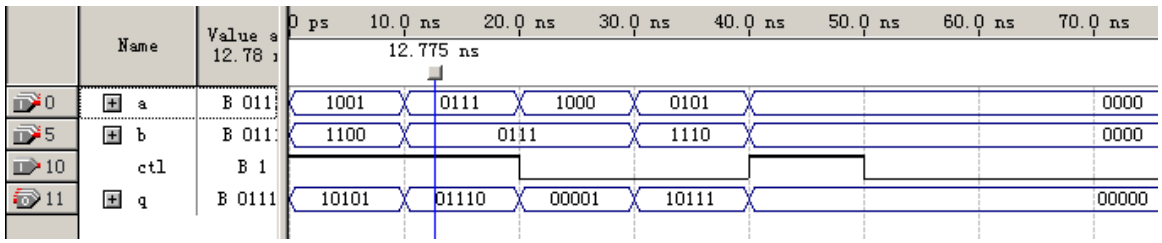


3、建立仿真文件

点击主工具栏上的 图标进行半编译，完成后新建一个波形仿真文件 File→New→Verification/Debugging Files→Vector Vaveform File.,然后在左边空白处双击左键添加仿真管脚。完成后添加激励信号的波形，首先使用鼠标选中一个信号（变蓝），利用左边工具添加仿真信号，根据需要依次添加各个信号（可以根据老师的实际要求添加不同的信号）,并保存。

4、功能仿真

选择 Processing→Simulator Tool 在弹出对话框中在仿真模式中选择”Functional” 然后点击”Generate Functional Simulation Netlist”生成功能仿真的 Netlist，完成后点击”Start”按钮开始仿真，完成后点击”Report”来查看仿真结果,如图所示。



5、分配管脚，下载、连线

选择 Assignments →Pins Planner 在 Location 中选择要分配的管脚，分配完毕后，点击主工具

栏的 图标进行全编译，完成后点击 下载到目标器件。连线时 a(3..0) 、b(3..0)、ctl 分配的管脚连接拨码开关，q(4..0)所对应管脚连接 LED 指示灯。

6、实验记录

根据仿真结果和实验 led 发光二极管的亮灭完成下表

输入			实验结果
Ctl	Ai（3..0）	Bi（3..0）	q（4..0）LED
0	1000	0111	
0	0101	1110	
1	1001	1100	
1	0111	0111	

管脚分配：

Pin182—a0 pin184—a1 pin186—a2 pin188—a3

Pin194—b0 pin196—b1 pin198—b2 pin200—b3

Pin202—q0 pin207—q1 pin216—q2 pin218—q3 pin220—a4

Pin181—ctl

实验六　组合逻辑 3-8 译码器的设计

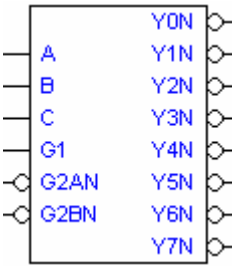
一、实验目的

设计并实现一个组合逻辑 3-8 译码器。

二、实验原理

常用的译码器有：2-4 译码器、3-8 译码器、4-16 译码器，下面我们用一个 3-8 译码器的设计来介绍译码器的设计方法。

3-8 译码器如下图所示：



3-8 译码器真值表

输入						输出							
G1	G2A	G2B	A	B	C	Y0N	Y1N	Y2N	Y3N	Y4N	Y5N	Y6N	Y7N
0	X	X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

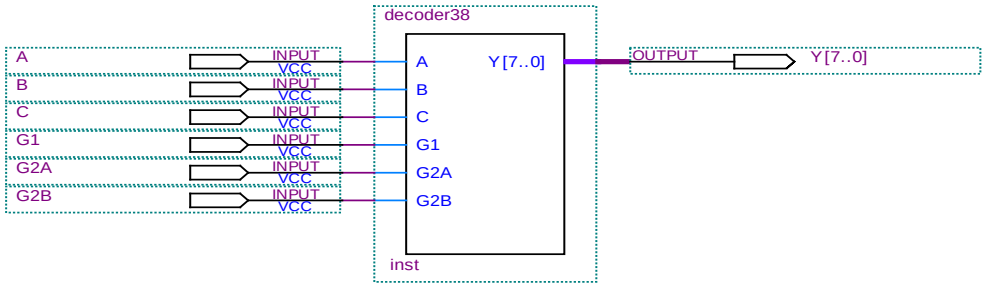
三、实验步骤

1、建立工程 encode，新建 VHDL 文件输入以下代码保存为 encode38.vhdl。打开 encode38.vhdl 文件，选择 File→ Creat/Update→ Creat Symbol Files for Current Files 生成顶层符号文件。（参考实验一、二）

LIBRARY IEEE;

```
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY decoder38 IS
    PORT(
        A, B,C,G1,G2A,G2B:   IN   STD_LOGIC;
        Y: OUT   STD_LOGIC_VECTOR(7 DOWNTO 0));
END decoder38;
ARCHITECTURE fun OF decoder38 IS
    SIGNAL indata: STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    indata <= C&B&A;
encoder:
    PROCESS (indata, G1, G2A,G2B)
    BEGIN
        IF (G1='1' AND G2A='0' AND G2B='0') THEN
            CASE indata IS
                WHEN "000"=>Y<="11111110";
                WHEN "001"=>Y<="11111101";
                WHEN "010"=>Y<="11111011";
                WHEN "011"=>Y<="11110111";
                WHEN "100"=>Y<="11101111";
                WHEN "101"=>Y<="11011111";
                WHEN "110"=>Y<="10111111";
                WHEN "111"=>Y<="01111111";
                WHEN OTHERS =>Y<="XXXXXXXX";
            END CASE;
        ELSE
            Y<="11111111";
        END IF;
    END PROCESS encoder;
    END fun;
```

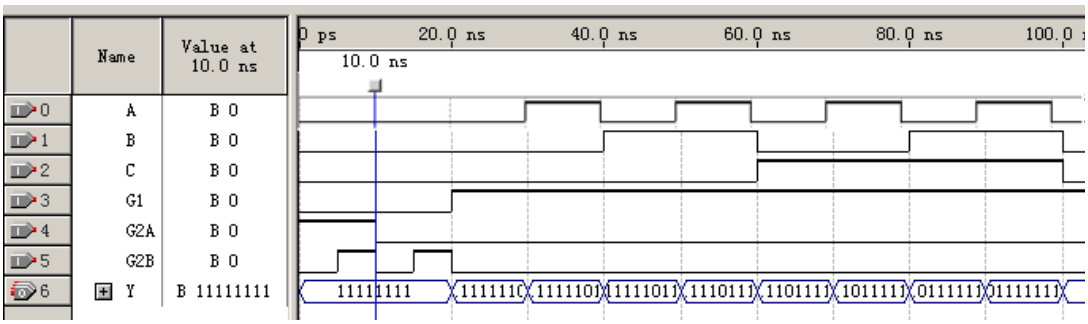
2、新建一个 Block Diagramm/Schematic File 原理图文件，，在空白处双击添加生成的顶层原理图，并连接 input、output（输入输出管脚）重命名后如图所示，保存文件 encoder38.bdf。



3、点击主工具栏上的 图标进行半编译，完成后新建一个波形仿真文件 File→New→Verification/Debugging Files→Vector Vaveform File.,然后在左边空白处双击左键添加仿真管脚。完成后添加激励信号的波形，首先使用鼠标选中一个信号（变蓝），利用左边工具添加仿真信号，根据需要依次添加各个信号（可以根据老师的实际要求添加不同的信号），并保存。

4、功能仿真

选择 Processing→Simulator Tool 在弹出对话框中在仿真模式中选择”Functional” 然后点击”Generate Functional Simulation Netlist”生成功能仿真的 Netlist，完成后点击”Start”按钮开始仿真，完成后点击”Report”来查看仿真结果，如图所示。



5、分配管脚，下载、连线

选择 Assignments →Pins Planner 在 Location 中选择要分配的管脚，分配完毕后，点击主工具栏的 图标进行全编译，完成后点击 下载到目标器件。连线时 A、B、C、G1、G2 分配的管脚连接拨码开关，Y(7..0)所对应管脚连接 LED 发光二极管。

6、实验记录

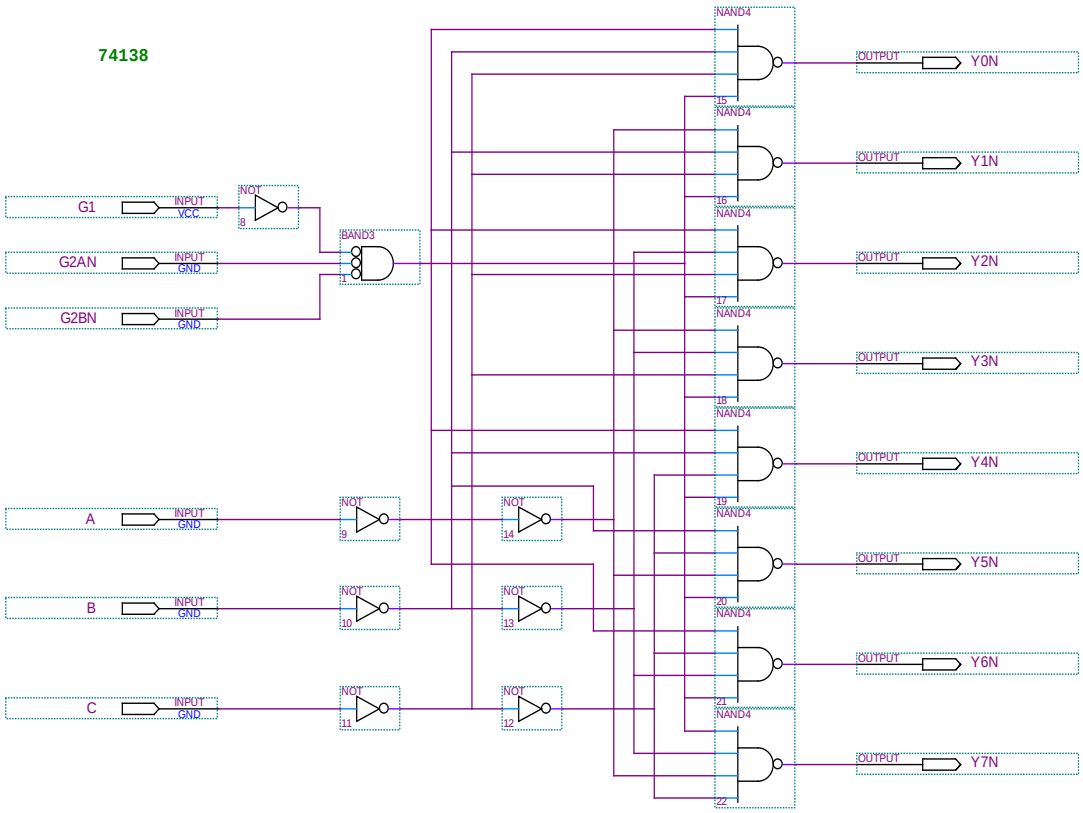
根据仿真结果和实验 led 发光二极管的亮灭完成下表，并分析其运算结果的正确性。

输入						输出							
G1	G2A	G2B	A	B	C	LED1	LED2	LED3	LED4	LED5	LED6	LED7	LED8
0	X	X	X	X	X								
1	0	0	0	0	0								
1	0	0	0	0	1								
1	0	0	0	1	0								
1	0	0	0	1	1								
1	0	0	1	0	0								

1	0	0	1	0	1								
1	0	0	1	1	0								
1	0	0	1	1	1								

附： 3-8 译码器的原理图设计

按照实验一的方法直接建立原理图文件也可以连接成38译码器,然后保存成默认的工程名即可。



管脚分配：

Pin182—A pin184—B pin186—C pin181—G1 pin183—G2A pin185—G2B

Pin216—Y0 pin207—Y1 Pin202—Y2 Pin200—Y3 Pin198—Y4 Pin196—Y5

Pin194—Y6 Pin188—Y7

实验七　数据比较器

一、实验目的

设计并实现一个 4 位二进制数据比较器。

二、实验原理

二进制比较器是提供关于两个二进制操作数间关系信息的逻辑电路。两个操作数的比较结果有三种情况：A 等于 B、A 大于 B 和 A 小于 B。

考虑当操作数 A 和 B 都是一位二进制数时，构造比较器的真值表见下表。

输入		输出		
a	b	eq（A=B）	hi(A>B)	lo(A<B)
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

在一位比较器的基础上，我们可以继续得到两位比较器，然后通过“迭代设计”得到 4 位的数据比较器。对于 4 位比较器的设计，我们可以通过原理图输入法或 VHDL 描述来完成，其中用 VHDL 语言描述是一种最为简单的方法。

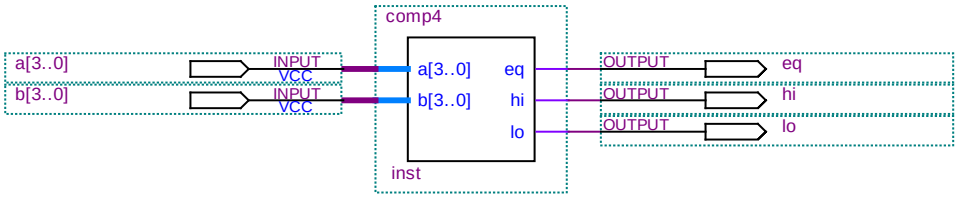
二、实验步骤

1、建立工程 comp，新建 VHDL 文件输入以下代码保存为 comp4.vhdl。打开 comp4.vhdl 文件，选择 File→ Creat/Update→ Creat Symbol Files for Current Files 生成顶层符号文件（参考实验一、二）。

```
library ieee;
use ieee.std_logic_1164.all;
entity comp4 is
   port(a,b:in std_logic_vector(3 downto 0);
        eq,hi,lo:out std_logic);
end comp4;
architecture comp of comp4 is
begin
process(a,b)
```

```
    begin
      if (a>b) then
         eq<='0'; hi<='1';lo<='0';
      elsif (a<b) then
            eq<='0';hi<='0';lo<='1';
      elsif (a=b) then
             eq<='1';hi<='0';lo<='0';
      else     eq<='0';hi<='0';lo<='0';
   end if;
   end process;
   end comp;
```

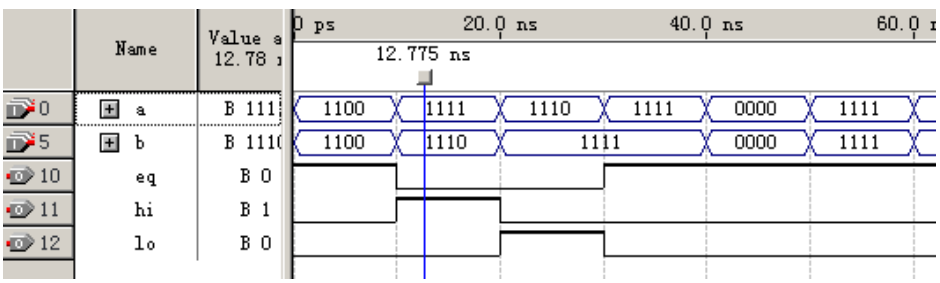
2、然后新建一个 Block Diagramm/Schematic File 原理图文件，在空白处双击添加生成的顶层原理图，并连接 input、output（输入输出管脚）重命名后如图所示，保存文件 comp.bdf。



3、点击主工具栏上的 图标进行半编译，完成后新建一个波形仿真文件 File→New→Verification/Debugging Files→Vector Vaveform File.,然后在左边空白处双击左键添加仿真管脚。完成后添加激励信号的波形，首先使用鼠标选中一个信号（变蓝），利用左边工具添加仿真信号，根据需要依次添加各个信号（可以根据老师的实际要求添加不同的信号），并保存。

4、功能仿真

选择 Processing→Simulator Tool 在弹出对话框中在仿真模式中选择”Functional” 然后点击”Generate Functional Simulation Netlist”生成功能仿真的 Netlist，完成后点击”Start”按钮开始仿真，完成后点击”Report”来查看仿真结果，如图所示。



5、分配管脚，下载、连线

选择 Assignments →Pins Planner 在 Location 中选择要分配的管脚，分配完毕后，点击主工具栏的 图标进行全编译，完成后点击 下载到目标器件。

连线时输入信号 a0~a3、b0~b3 所分配的管脚分别连接拨码开关，输出信号 eq、hi、lo 所对应管脚分别连接 LED 发光二极管。

6、实验记录

根据仿真结果和实验 led 发光二极管的亮灭完成下表，并分析其运算结果的正确性。

输入		输出		
A0~A3	B0~B3	eq（LED1）	hi(LED2)	lo(LED3)
0000	0000			
1100	1100			
1110	1111			
1111	1110			
1111	1111			

管脚分配：

Pin182—a0 pin184—a1 pin186—a2 pin188—a3

Pin194—b0 pin196—b1 pin198—b2 pin200—b3

Pin202—eq pin207—hi pin216—lo

实验八　多路数据选择器

一、实验目的

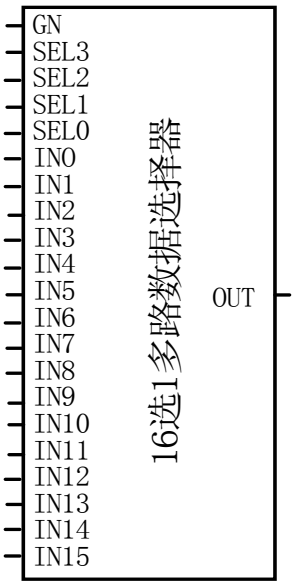
设计并实现一个 16 选 1 多路数据选择器。

二、实验原理

在 VHDL 语言中描述一个 2 选一的多路选择器的方法有多种，例如：在一个进程中使 if-when-else 语句；在一个进程中 case 语句；使用 with select 构造或使用结构 VHDL。推荐的使用方法是使用 when else 构造，这样在 VHDL 代码中只用 1 行就可以描述 2 选 1 多路选择器。例如：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MUX2 IS
    PORT(A,B,SEL:    IN    STD_LOGIC;
                 Q:   OUT       STD_LOGIC);
END;
ARCHITECTURE A OF MUX2 IS
BEGIN
    Q <= A    WHEN    SEL='0'    ELSE    B;
END;
```

但我们在描述一个 16 选 1 的多路选择器时，若采用同样的方法，则就需要许多行 VHDL 代码，此时我们可以在进程中使用 case 语句会很清晰，但无论使用哪一种描述方法，综合得到的结果是相同的。16 选 1 多路选择器结构图如下图所示。



三、实验步骤

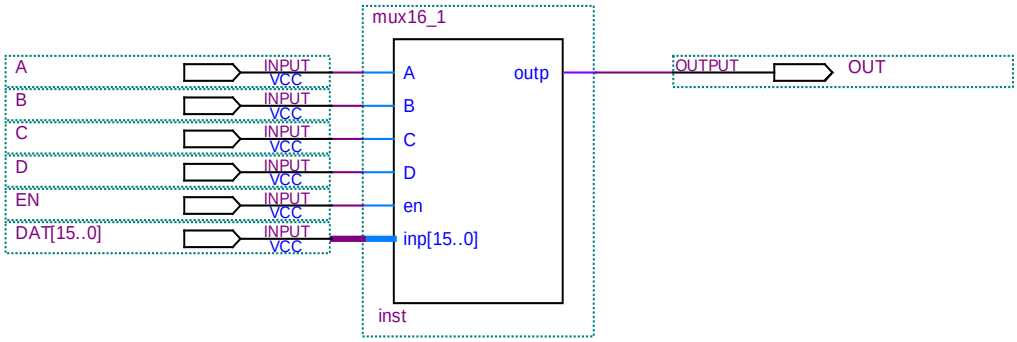
1、建立工程 select，新建 VHDL 文件输入以下代码保存为 mux16_1.vhdl。打开 comp4.vhdl

文件，选择 File→ Creat/Update→ Creat Symbol Files for Current Files 生成顶层符号文件。（参考实验一、二）

```
library ieee;
use ieee.std_logic_1164.all;
entity mux16_1 is
   port(
        A,B,C,D:in std_logic;-----数据选择端
        en        : in std_logic;
        inp:in std_logic_vector (15 downto 0);--数据输入
        outp:out std_logic);----------------------数据输出
end mux16_1;
architecture tst of mux16_1 is
  signal sel: std_logic_vector (3 downto 0);
begin
      sel<=D&C&B&A;
process(en,sel,inp)
   begin
    if(en='1') then
         case sel is
             when"0000"=>outp<=inp(0);
             when"0001"=>outp<=inp(1);
             when"0010"=>outp<=inp(2);
             when"0011"=>outp<=inp(3);
             when"0100"=>outp<=inp(4);
             when"0101"=>outp<=inp(5);
             when"0110"=>outp<=inp(6);
             when"0111"=>outp<=inp(7);
             when"1000"=>outp<=inp(8);
             when"1001"=>outp<=inp(9);
             when"1010"=>outp<=inp(10);
             when"1011"=>outp<=inp(11);
             when"1100"=>outp<=inp(12);
             when"1101"=>outp<=inp(13);
             when"1110"=>outp<=inp(14);
             when"1111"=>outp<=inp(15);
        end case;
   else
```

```
        outp<='0';
    end if;
    end process;
end tst;
```

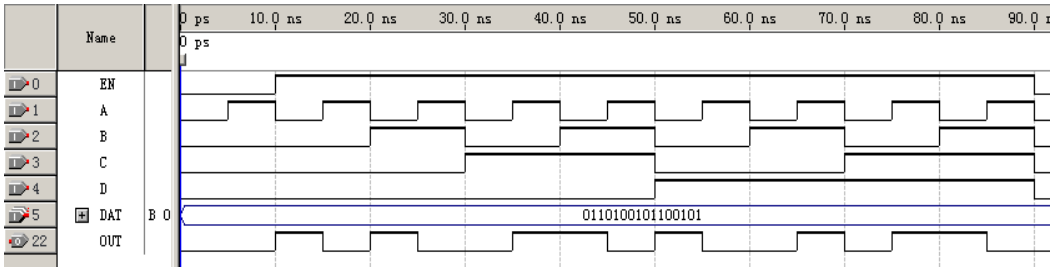
2、 然后新建一个 Block Diagramm/Schematic File 原理图文件，在空白处双击添加生成的顶层原理图，并连接 input、output（输入输出管脚）重命名后如图所示，保存文件 select.bdf。）



3、点击主工具栏上的 图标进行半编译，完成后新建一个波形仿真文件 File→New→Verification/Debugging Files→Vector Vaveform File.,然后在左边空白处双击左键添加仿真管脚。完成后添加激励信号的波形，首先使用鼠标选中一个信号（变蓝），利用左边工具添加仿真信号，根据需要依次添加各个信号（可以根据老师的实际要求添加不同的信号）。 例如这里 16 位数据管脚赋值”0110100101100101”，并保存，可以是系统默认的名称。

4、功能仿真

选择 Processing→Simulator Tool 在弹出对话框中在仿真模式中选择”Functional” 然后点击”Generate Functional Simulation Netlist”生成功能仿真的 Netlist，当有多个仿真文件时可以点击 Add Multiple Files... 来添加其他的文件,不需要时则无需操作。完成后点击”Start”按钮开始仿真，完成后点击”Report”来查看仿真结果，如图所示。



5、分配管脚，下载、连线

选择 Assignments →Pins Planner 在 Location 中选择要分配的管脚，分配完毕后，点击主工具栏的 图标进行全编译，完成后点击 下载到目标器件。

连线时输入信号 EN、A、B、C、D、DAT(15..0)所分配的管脚分别连接拨码开关，输出信号 OUT 所对应管脚连接 LED 发光二极管。

6、实验记录

根据仿真结果和实验 led 发光二极管的亮灭完成下表，并分析其运算结果的正确性。并自行完成 32 选一的多路数据选择器。

输入信号						输出
EN	A	B	C	D	DAT(15..0)	OUT (LED)
0	X	X	X	X		0
1	1	0	0	0		
1	0	1	0	0		
1	1	1	0	0		
1	0	0	1	0		
1	1	0	1	0		
1	0	1	1	0		
1	1	1	1	0		
1	0	0	0	1	0110100101100101	
1	1	0	0	1		
1	0	1	0	1		
1	1	1	0	1		
1	0	0	1	1		
1	1	0	1	1		
1	0	1	1	1		
1	1	1	1	1		
1	0	0	0	0		

管脚分配：

Pin181—A pin183—B pin185—C pin187—D pin189—en pin195—outp

Pin182—inp0 pin184—inp1 pin186—inp2 pin188—inp3 pin194—inp4 pin196—inp5

Pin198—inp6 pin200—inp7 pin202—inp8 pin207—inp9 pin216—inp10 pin218—inp11

Pin220—inp12 pin222—inp13 pin224—inp14 pin230—inp15

实验九　编码器

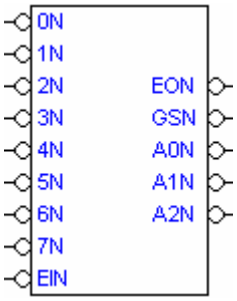
一、实验目的

设计并实现一个 16-4 优先编码器。

二、实验原理

常用的编码器有：4-2 编码器、8-3 编码器、16-4 编码器，编码器的逻辑功能就是把输入的每一个高低电平信号变成一个对应的二进制代码。在优先编码器电路中，允许同时输入两个以上编码信号。当几个输入信号同时出现时，只对其中优先权最高的一个进行编码。下面我们用一个 8-3 编码器的来介绍编码器的原理方法。

8-3 编码器如图所示：



8-3 优先编码器真值表

输入									输出				
EIN	0N	1N	2N	3N	4N	5N	6N	7N	A2N	A1N	A0N	GSN	EON
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	X	X	X	X	X	X	X	0	0	0	0	0	1
0	X	X	X	X	X	X	0	1	0	0	1	0	1
0	X	X	X	X	X	0	1	1	0	1	0	0	1
0	X	X	X	X	0	1	1	1	0	1	1	0	1
0	X	X	X	0	1	1	1	1	1	0	0	0	1
0	X	X	0	1	1	1	1	1	1	0	1	0	1
0	X	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1

三、实验步骤

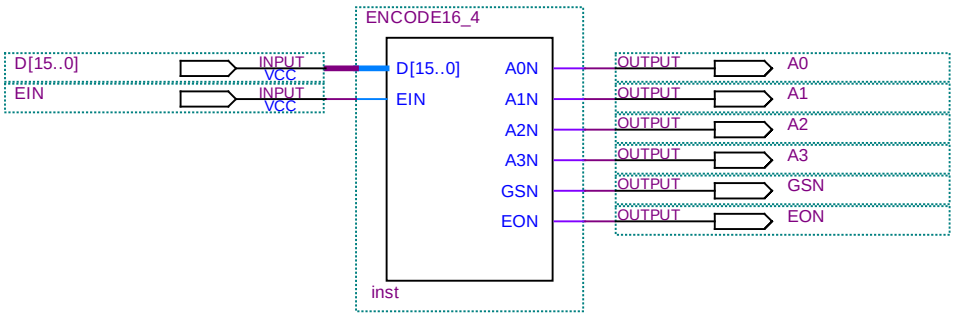
1、建立工程 encode，新建 VHDL 文件输入以下代码保存为 ENCODE16_4.vhdl。打开

ENCODE16_4.vhdl 文件，选择 File→ Creat/Update→ Creat Symbol Files for Current Files 生成顶层符号文件。（参考实验一、二）

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY    ENCODE16_4 IS
   PORT(D: IN    STD_LOGIC_VECTOR(15 DOWNTO 0);
        EIN:IN    STD_LOGIC;---
        A0N,A1N,A2N,A3N,GSN,EON:   OUT STD_LOGIC);
END ENCODE16_4;
ARCHITECTURE A OF ENCODE16_4 IS
SIGNAL Q: STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
A0N <=Q(0);    A1N<=Q(1);    A2N<=Q(2); A3N<=Q(3);
   PROCESS(D)
   BEGIN
        IF EIN ='1'THEN
             Q<="1111";
             GSN<='1';    EON<='1';
        ELSIF     D(0)='0'   THEN
             Q<="1111";    GSN<='0';      EON<='1';
        ELSIF     D(1)='0'   THEN
             Q<="1110";    GSN<='0';      EON<='1';
        ELSIF     D(2)='0'   THEN
             Q<="1101";    GSN<='0';      EON<='1';
        ELSIF     D(3)='0'   THEN
             Q<="1100";    GSN<='0';      EON<='1';
        ELSIF     D(4)='0'   THEN
             Q<="1011";    GSN<='0';      EON<='1';
        ELSIF     D(5)='0'   THEN
             Q<="1010";    GSN<='0';      EON<='1';
        ELSIF     D(6)='0'   THEN
             Q<="1001";    GSN<='0';      EON<='1';
        ELSIF     D(7)='0'   THEN
             Q<="1000";    GSN<='0';      EON<='1';
        ELSIF     D(8)='0'   THEN
             Q<="0111";    GSN<='0';      EON<='1';
        ELSIF     D(9)='0'   THEN
             Q<="0110";    GSN<='0';      EON<='1';
```

```
        ELSIF   D(10)='0' THEN
            Q<="0101";   GSN<='0';    EON<='1';
        ELSIF   D(11)='0' THEN
            Q<="0100";   GSN<='0';    EON<='1';
        ELSIF   D(12)='0' THEN
            Q<="0011";   GSN<='0';    EON<='1';
        ELSIF   D(13)='0' THEN
            Q<="0010";   GSN<='0';    EON<='1';
        ELSIF   D(14)='0' THEN
            Q<="0001";   GSN<='0';    EON<='1';
        ELSIF   D(15)='0' THEN
            Q<="0000";   GSN<='0';    EON<='1';
        ELSIF   D="1111111111111111"      THEN
            Q<="1111";   GSN<='1';    EON<='1';
        END IF;
    END PROCESS;
END A;
```

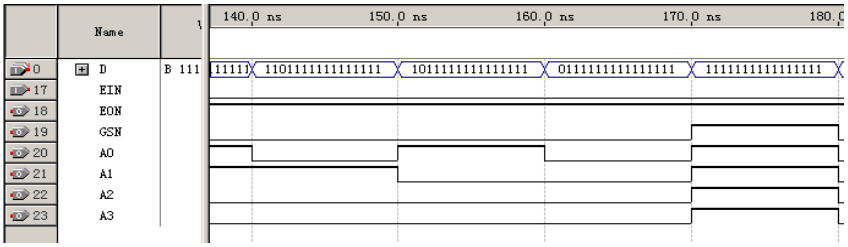
2、新建一个 Block Diagramm/Schematic File 原理图文件，在空白处双击添加生成的顶层原理图，并连接 input、output（输入输出管脚）重命名后如图所示，保存文件 encode.bdf。



3、点击主工具栏上的 图标进行半编译，完成后新建一个波形仿真文件 File→New→Verification/Debugging Files→Vector Vaveform File.,然后在左边空白处双击左键添加仿真管脚。完成后添加激励信号的波形，首先编码信号使用鼠标选中一个信号的区域（变蓝），利用左键双击选中区域输入二进制码，其他根据需要依次添加各个信号（可以根据老师的实际要求添加不同的信号）。并保存，可以是系统默认的名称。

4、功能仿真

选择 Processing→Simulator Tool 在弹出对话框中在仿真模式中选择”Functional” 然后点击”Generate Functional Simulation Netlist”生成功能仿真的 Netlist，完成后点击”Start”按钮开始仿真，完成后点击”Report”来查看仿真结果，如下图所示为截取的部分仿真波形。



5、分配管脚，下载、连线

选择 Assignments →Pins Planner 在 Location 中选择要分配的管脚，分配完毕后，点击主工具栏的 进行全编译，完成后点击 下载到目标器件。连线时输入信号 EIN、D(15..0)所分配的管脚分别连拨码开关，输出信号 GSN、EON、A0、A1、A2、A3 连接 LED 发光二极管。

6、实验记录

根据仿真结果和实验 led 发光二极管的亮灭完成下表，并分析其运算结果的正确性。

输入 输出 LED

EIN	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	A0	A1	A2	A3	GS	EO
1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X						
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1						
0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X						
0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X						
0	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X						
0	1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X						
0	1	1	1	1	0	X	X	X	X	X	X	X	X	X	X	X						
0	1	1	1	1	1	0	X	X	X	X	X	X	X	X	X	X						
0	1	1	1	1	1	1	0	X	X	X	X	X	X	X	X	X						
0	1	1	1	1	1	1	1	0	X	X	X	X	X	X	X	X						
0	1	1	1	1	1	1	1	1	0	X	X	X	X	X	X	X						
0	1	1	1	1		1	1	1	1	0	X	X	X	X	X	X						
0	1	1	1	1	1	1	1	1	1	1	0	X	X	X	X	X						
0	1	1	1	1	1	1	1	1	1	1	1	0	X	X	X	X						
0	1	1	1	1	1	1	1	1	1	1	1	1	0	X	X	X						
0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	X	X						
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	X						

管脚分配：

Pin181—A0N pin183—A1N pin185—A2N pin187—A3N

pin189—EIN pin195—EON PIN197--GSN

Pin182—D0 pin184—D1 pin186—D2 pin188—D3 pin194—D4 pin196—D5

Pin198—D6 pin200—D7 pin202—D8 pin207—D9 pin216—D10 pin218—D11

Pin220—D12 pin222—D13 pin224—D14 pin230—D15

实验十　计数器

一、实验目的

1、设计一个增 1 计数器；

2、设计一个带使能输入及同步清 0 的增 1/减 1 的 3 位计数器。

二、实验原理

在用 VHDL 语言描述一个计数器时，如果使用了程序包 ieee.std_logic_unsigned，则在描述计数器时就可以使用其中的函数“+”（递增计数）和“-”（递减计数）。假定设计对象是增 1 计数器并且计数器被说明为向量，则当所有位均为‘1’时，计数器的下一状态将自动变成‘0’。举例来说，假定计数器的值到达“111”是将停止，则在增 1 之前必须测试计数器的值。

如果计数器被说明为整数类型，则必须有上限值测试。否则，在计数值等于 7，并且要执行增 1 操作时，模拟器将指出此时有错误发生。

三、实验步骤

1、三位增一计数器

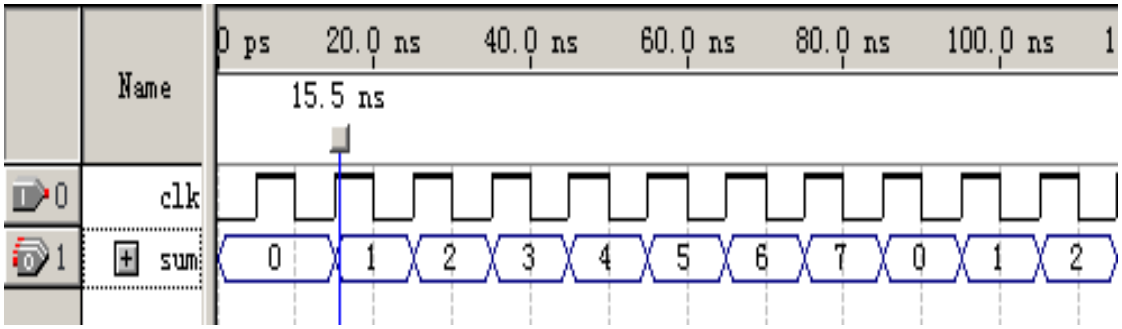
参考程序、顶层文件图及仿真结果如下图所示。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity plus is
port(
      clk: in std_logic;
      sum: out std_logic_vector(0 downto 2)
    );
end plus;

architecture arcount of plus is
signal temp: std_logic_vector(0 downto 2);
begin
process(clk)
begin
```

```
if(clk='1' and clk'event) then
temp<=temp+1;
sum<=temp;
end if;
end process;
end arcount;
```

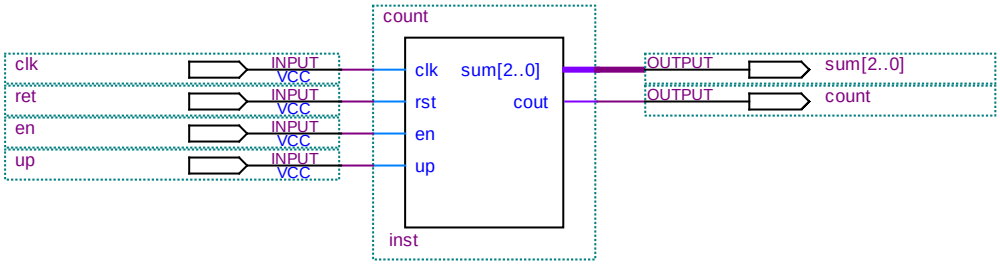


2、带使能输入及同步清 0 的增 1/减 1 的 3 位计数器

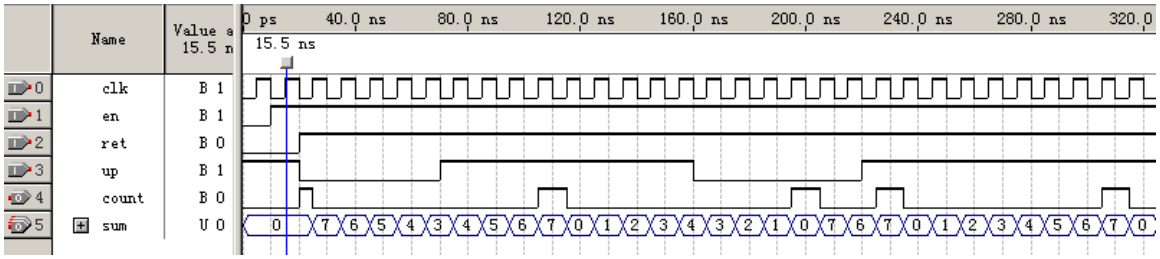
建立工程并生成原理图文件如下图所示。其中 CLK 接按键输入，其他输入引脚接拨码开关，输出引脚接 LED 发光二极管，半编译后功能仿真。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity count is
port(clk,rst,en,up:in std_logic;
     sum: out std_logic_vector(2 downto 0 );
     cout:out std_logic);
end;
architecture A of count is
signal cou: std_logic_vector(2 downto 0 );
begin
  process(clk,rst)
begin
if rst='0' then
    cou<=(others=>'0');
```

```
elsif rising_edge(clk)then
  if en='1'then
 case up is
   when '1'=> cou<=cou+1;
  when others=> cou<=cou-1;
 end case;
 end if;
 end if;
end process;
sum<= cou;
cout<='1'when en='1' and ((up='1' and cou=7)or( up='0' and cou=0))else'0';
end A;
```



仿真结果：



3、实验记录

根据仿真结果分析实验现象。自己思考完成一个把 clk256 分频的时钟输出。

管脚分配：

Pin31—clk　pin209—rst　pin181—en　pin183—up　pin185—count

Pin182—sum0　pin184—sum1　pin186—sum2

实验十一 7 段数码管控制接口

一、实验目的

1、学习 7 段数码显示译码器设计；

2、巩固译码器的设计；

3、学习多层次设计方法。

二、实验原理

本实验板有两个共阳数码管，彼此独立显示。其 0-9,A-F 对应的共阳码分别为 0xC0, 0XF9, 0xA4, 0xB0, 0x99, 0x92,0x82,0xF8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e。

实验设计四个拨码开关在”0000” 到 ”1111”变换实现对数码管 0-9 及 A-F 的编码。并在两个数码管上同时显示。

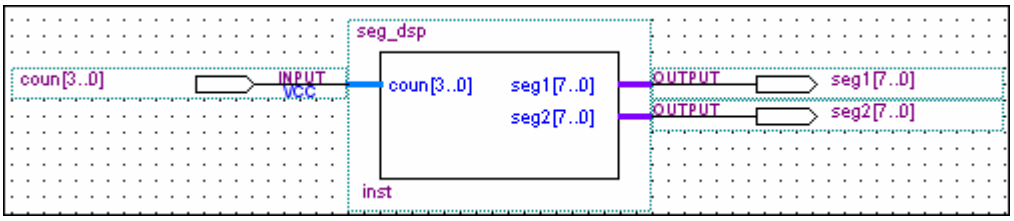
三、实验步骤

1、建立工程 segdisp，新建 VHDL 文件输入以下代码保存为 seg_dsp.vhdl。打开 seg_dsp.vhdl 文件，选择 File→ Creat/Update→ Creat Symbol Files for Current Files 生成顶层符号文件。（参考实验一、二）

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity seg_dsp is
   port(
         coun : in std_logic_vector(3 downto 0)；
         seg1 :out std_logic_vector(7 downto 0);
         seg2 :out std_logic_vector(7 downto 0)
       );
end seg_dsp;
architecture test of seg_dsp is
   signal seg:std_logic_vector(7 downto 0);
begin
process(coun)
begin
```

```
        case   coun   is
        when "0000"=>seg<="11000000" ;--0xc0
        when "0001"=>seg<="11111001" ;--0xf9
        when "0010"=>seg<="10100100" ; --0xa4
        when "0011"=>seg<="10110000" ;--0xb0
        when"0100"=> seg<="10011001"   ;--0x99
        when"0101"=> seg<="10010010";--0x92
        when"0110"=>seg <="10000010"   ;--0x82
        when "0111"=> seg<="11111000" ;--0xf8
        when "1000"=> seg<="10000000" ;--0x80
        when "1001"=>seg<="10010000"   ;--0x90
        when "1010"=>seg<="10001000" ;--0x88
        when "1011"=>seg<="10000011" ;--0x83
        when "1100"=>seg<="10100110"   ;--0xc6
        when "1101"=>seg<="10100001"   ;--0xa1
        when "1110"=>seg<="10000110"   ;--0x86
        when "1111"=>seg<="10001110"   ;--0x8e
        when others=>seg<="00000000";
        end case;
        seg1<=seg;
        seg2<=seg;
    end process;
    end test;
```

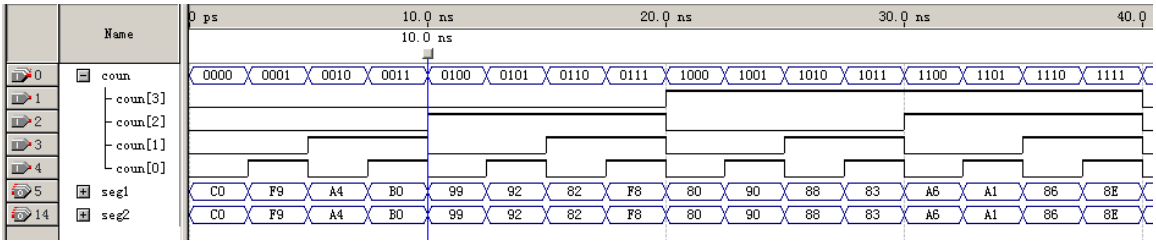
2、然后新建一个 Block Diagramm/Schematic File 原理图文件，在空白处双击添加生成的顶层原理图，并连接 input、output（输入输出管脚）重命名后如下图所示。保存文件 seg_disp.bdf。



3、点击主工具栏上的 图标进行半编译，完成后新建一个波形仿真文件 File→New→Verification/Debugging Files→Vector Vaveform File.,然后在左边空白处双击左键添加仿真管脚。完成后添加激励信号的波形，首先编码信号使用鼠标选中一个信号的区域（变蓝），利用左键双击选中区域输入二进制码，其他根据需要依次添加各个信号（可以根据老师的实际要求添加不同的信号）输出信号选种后右键可以修改器显示属性为 HEX（十六进制）显示。并保存，可以是系统默认的名称。

4、功能仿真

选择 Processing→Simulator Tool 在弹出对话框中在仿真模式中选择”Functional” 然后点击”Generate Functional Simulation Netlist”生成功能仿真的 Netlist，完成后点击”Start”按钮开始仿真，完成后点击”Report”来查看仿真结果，如图所示为截取的部分仿真波形。



5、分配管脚，下载、连线

选择 Assignments →Pins Planner 在 Location 中选择要分配的管脚，分配完毕后，点击主工具栏的 图标进行全编译，完成后点击 下载到目标器件。

连线时输入信号 coun(3..0)所分配的管脚分别连接拨码开关，输出信号 seg1[7..0]、seg2[7..0]所对应管脚分别连接两个数码管所对应的 dp、g、f、e、d、c、b、a。

6、实验记录

根据仿真结果和实验时数码管的显示完成下表，把数码管显示的字符填入下表，并分析其运算结果的正确性。

IN	Coun	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
OUT	显示																

管脚分配：

set_location_assignment PIN_182 -to coun[0]

set_location_assignment PIN_184 -to coun[1]

set_location_assignment PIN_186 -to coun[2]

set_location_assignment PIN_188 -to coun[3]

```
set_location_assignment PIN_194 -to seg1[0]
set_location_assignment PIN_196 -to seg1[1]
set_location_assignment PIN_198 -to seg1[2]
set_location_assignment PIN_200 -to seg1[3]
set_location_assignment PIN_202 -to seg1[4]
set_location_assignment PIN_207 -to seg1[5]
set_location_assignment PIN_216 -to seg1[6]
set_location_assignment PIN_218 -to seg1[7]
set_location_assignment PIN_220 -to seg2[0]
set_location_assignment PIN_222 -to seg2[1]
set_location_assignment PIN_224 -to seg2[2]
set_location_assignment PIN_230 -to seg2[3]
set_location_assignment PIN_232 -to seg2[4]
set_location_assignment PIN_234 -to seg2[5]
set_location_assignment PIN_236 -to seg2[6]
set_location_assignment PIN_238 -to seg2[7]
```

实验十二 16*16 点阵设计

一、实验目的

1、学习点阵式 LED 显示的原理；

2、学习动态扫描显示的原理；

3、学习设计多字符点阵式;LED 显示控制系统的设计方法。

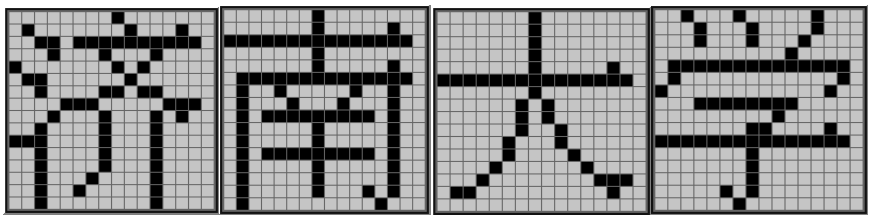
二、实验原理

点阵是将 LED 显示二极管按照行列的顺序排列在一起，形成一个显示单元，常用的点阵式 LED 显示器是采用 8*8 点阵结构，一个显示单元可以显示一个 ASCII 字符，四个点阵式 LED 可以组成一个 16*16 点阵结构，可以显示一个 16*16 点阵的汉字。

为驱动点阵式 LED 显示屏，通常情况是采用动态扫描方式，将点阵式 LED 构成的显示屏的同一行的 LED 短接在一起，控制电路一次输出每一行所对应的 LED 的亮灭状态，同时对应行输出有效，点亮该行对应的点，这样逐行显示，只要保证显示刷新频率大于 50Hz，就可以看到一个稳定的字符、汉字。

三、实验步骤

1、首先利用字模软件提取汉字的字模例如“济南大学”的字模，这里已经提取完毕，就不需要再次提取了。这里汉字的每一行由两个字节构成，例如“济“的第一行，左边字节为”00000000”即 0x00,右边字节为”1000”即 0x80,同样第三行为”0011011111111110”即 0x37、0xFE，其他同理。



“ 济 ” 0x00,0x80,0x40,0x44,0x37,0xFE,0x11,0x10,0x80,0xA0,0x60,0x40,0x21,0xB0,0x0E,0x0E,
0x11,0x14,0x21,0x10,0xE1,0x10,0x21,0x10, 0x21,0x10,0x22,0x10,0x20,0x10,0x24,0x10,

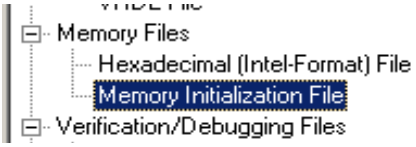
“ 南 ” 0x01,0x00,0x01,0x04,0xFF,0xFE,0x01,0x00, 0x01,0x04,0x7F,0xFE,0x48,0x24,0x44,0x44,
0x5F,0xF4,0x41,0x04,0x41,0x04,0x5F,0xF4, 0x41,0x04,0x41,0x04,0x41,0x14,0x40,0x08,

“ 大” 0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00, 0x01,0x04,0xFF,0xFE,0x01,0x00,0x02,0x80,
0x02,0x80,0x02,0x40,0x04,0x40,0x04,0x20, 0x08,0x10,0x10,0x0E,0x60,0x04,0x00,0x00,

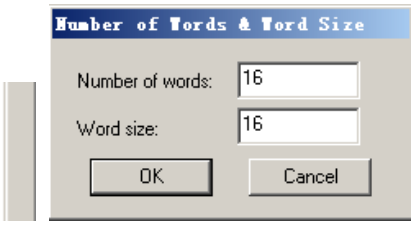
“ 学 ” 0x22,0x08,0x11,0x08,0x11,0x10,0x00,0x20, 0x7F,0xFE,0x40,0x02,0x80,0x04,0x1F,0xE0,
0x00,0x40,0x01,0x84,0xFF,0xFE,0x01,0x00, 0x01,0x00,0x01,0x00,0x05,0x00,0x02,0x00

2、建立工程 pots 文件,并新建字模数据表，将字模逐个写入。

建立工程后，新建 new→ Memory Files →memory initialization files →mif 文件

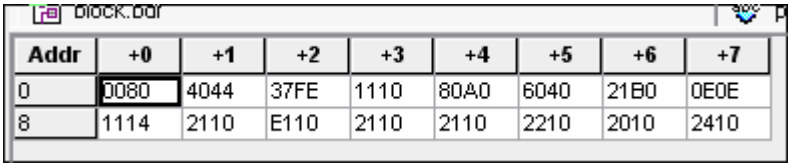


新建 MIF 数据表



设定数据表大小

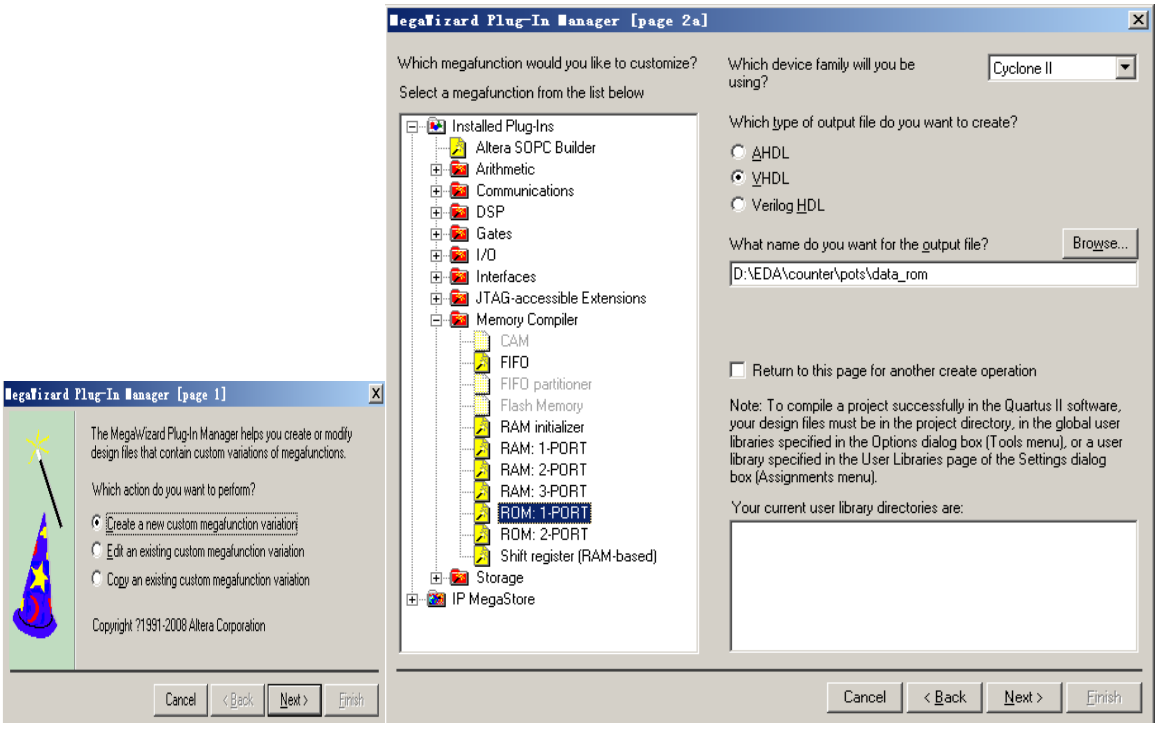
本实验以显示“济“字为例，说明点阵显示的方法，所以建立了宽度 16 位的 16 个数据表如图所示，然后将“济”字的字模依次输入，并保存命名为 data.mif，如下图所示。



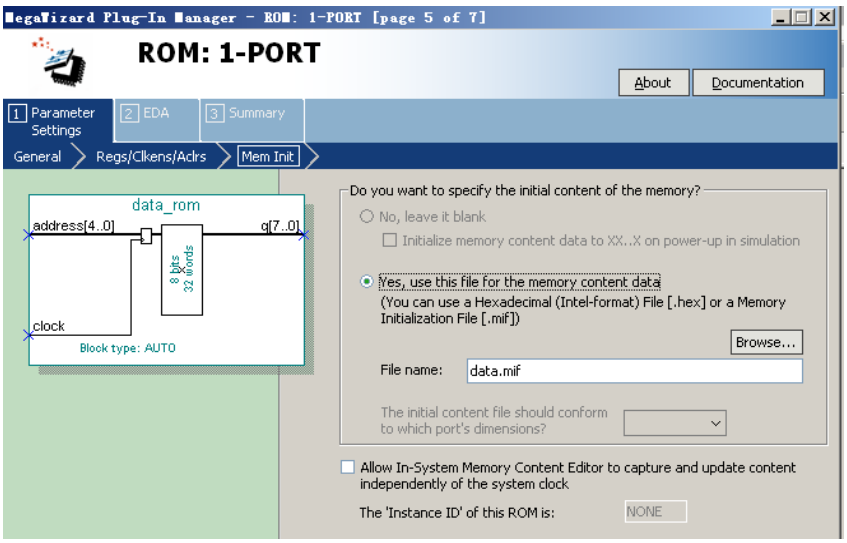
Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	0080	4044	37FE	1110	80A0	6040	21B0	0E0E
8	1114	2110	E110	2110	2110	2210	2010	2410

3、 建立 LPM_ROM 表

选择 Tools→Megawizard Plug-In Manager 弹出对话框如图所示。点击 next,选择 Memery complier→ROM:1-PORT，命名为 data_rom.，用于读 mif 文件接口。


MegaWizard Plug-In Manager [page 1]
The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.
Which action do you want to perform?
Create a new custom megafunction variation
Edit an existing custom megafunction variation
Copy an existing custom megafunction variation
Copyright ?1991-2008 Altera Corporation
Cancel
< Back
Next >
Finish
MegaWizard Plug-In Manager [page 2a]
Which megafunction would you like to customize?
Select a megafunction from the list below
Installed Plug-Ins
Altera SOPC Builder
Arithmetic
Communications
DSP
Gates
I/O
Interfaces
JTAG-accessible Extensions
Memory Compiler
CAM
FIFO
FIFO partitioner
Flash Memory
RAM initializer
RAM: 1-PORT
RAM: 2-PORT
RAM: 3-PORT
ROM: 1-PORT
ROM: 2-PORT
Shift register (RAM-based)
Storage
IP MegaStore
Which device family will you be using?
Cyclone II
Which type of output file do you want to create?
AHDL
VHDL
Verilog HDL
What name do you want for the output file?
Browse...
D:\EDA\counter\pots\data_rom
Return to this page for another create operation
Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the Options dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu).
Your current user library directories are:
Cancel
< Back
Next >
Finish

点击 Next，写入 16 位宽度的 16 个数据，再点击 Next 进入下图所示的界面，选择 data.mif 文件路径，来初始化 ROM 文件。

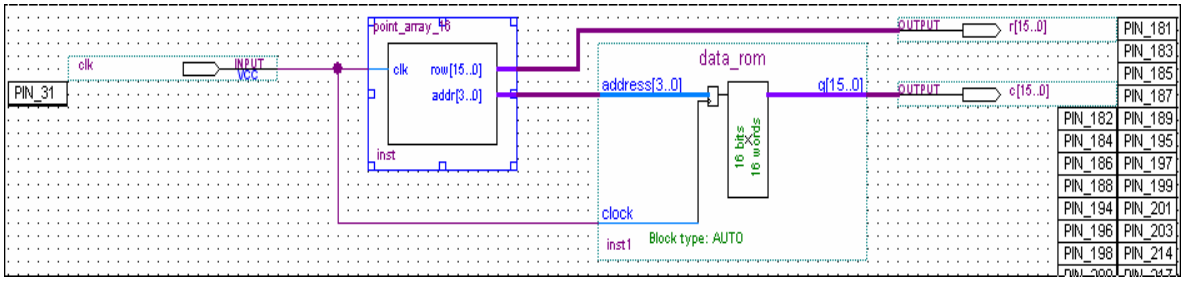

MegaWizard Plug-In Manager - ROM: 1-PORT [page 5 of 7]
ROM: 1-PORT
About
Documentation
1 Parameter Settings
2 EDA
3 Summary
General
Regs/Clkens/Aclrs
Mem Init
data_rom
address[4..0]
q[7..0]
8 bits
32 words
clock
Block type: AUTO
Do you want to specify the initial content of the memory?
No, leave it blank
Initialize memory content data to XX...X on power-up in simulation
Yes, use this file for the memory content data
(You can use a Hexadecimal (Intel-format) File [.hex] or a Memory Initialization File [.mif])
Browse...
File name:
data.mif
The initial content file should conform to which port's dimensions?
Allow In-System Memory Content Editor to capture and update content independently of the system clock
The 'Instance ID' of this ROM is:
NONE

4、新建 VHDL 文件输入以下代码保存为 point_array_16.vhdl。打开 point_array_16.vhdl 文件，选择 File→ Creat/Update→ Creat Symbol Files for Current Files 生成顶层符号文件。（参考实验一、二）

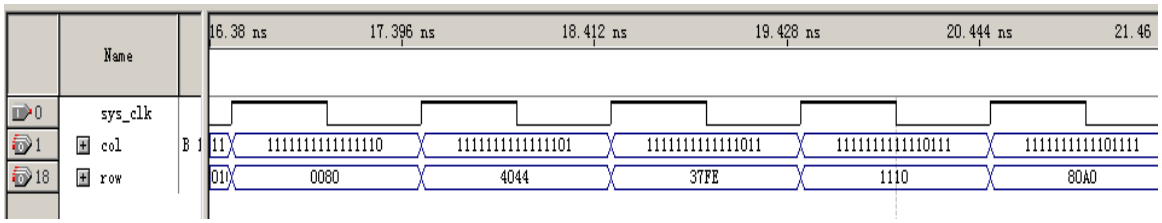
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity point_array_16 is
   port( clk:in std_logic;
         row:out std_logic_vector(15 downto 0);
         addr:out std_logic_vector(3 downto 0));
end point_array_16;
architecture aza of point_array_16 is
signal fre_clk  : std_logic;
signal addro: std_logic_vector(3 downto 0);
--------------------------------------------------------------------------------
--   COMPONENT data_rom     IS--调用数据存储器 LPM_ROM  文件：data_rom.vhd
--          PORT( address       : IN STD_LOGIC_VECTOR(3 DOWNTO 0); --4 位地址信号
--                clock   : IN STD_LOGIC ;--  地址锁时钟
--                q           : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
--   END COMPONENT;
--------------------------------------------------------------------------------
```

```
begin
a: process(clk)
   variable count: integer range 0 to 90;
      begin
      if(clk = '1' and clk'event) then
         count:=count+1;
                  if count<=10 then
                     fre_clk<='0';
                  elsif count>=20 then
                     count:=0;
                  else
                     fre_clk<='1';
                  end if;
                  end if;
     end process;
b: process(fre_clk)
     variable i : integer range 15 downto 0:=0;
     begin
     if(fre_clk'event and fre_clk='1') then
        row<="1111111111111111";
        addro<=addro+1;
        row(i+1)<='0';
        i:=i+1;
     end if;
     end process;
     addr<=addro;
end aza;
```

5、新建一个 Block Diagramm/Schematic File 原理图文件，在空白处双击添加生成的顶层原理图，并连接 input、output（输入输出管脚）重命名后如下图所示。保存文件 seg_disp.bdf。



6、功能仿真



7、分配管脚，下载、连线

选择 Assignments →Pins 在 Location 中选择要分配的管脚，其中系统时钟必须分配 pin-31 脚，其他引脚根据实验板上预留的插针管脚自由分配，完毕后，点击主工具栏的 图标进行全编译，完成后点击 下载到目标器件。

连线时输入信号 clk 已经连接，col（15..0）对应着 16 个行数据端，row(15..0)对应着 16 位的行选择端。

8、实验记录

进一步熟悉点阵的实现过程完成一下实验。

1）、改变 ROM 文件中的点阵数据，使用元件例化的方法，实现点阵”南”的静态显示。

例如：

```
COMPONENT data_rom     IS—声明调用数据存储器 LPM_ROM  文件：data_rom.vhd
PORT( address       : IN STD_LOGIC_VECTOR(3 DOWNTO 0); --4 位地址信号
        clock    : IN STD_LOGIC ;--  地址锁时钟
        q              : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
END COMPONENT;
--元件例化
u1:data_rom PORT MAP(address=>addr,q=>row,clock=>fre_clk);
```

2）、建立“济南大学“的 16*64 的 ROM 表，实现四个汉字的滚动显示。

管脚分配：

```
set_location_assignment PIN_182 -to c[0]
set_location_assignment PIN_184 -to c[1]
set_location_assignment PIN_186 -to c[2]
set_location_assignment PIN_188 -to c[3]
set_location_assignment PIN_194 -to c[4]
set_location_assignment PIN_196 -to c[5]
set_location_assignment PIN_198 -to c[6]
set_location_assignment PIN_200 -to c[7]
set_location_assignment PIN_202 -to c[8]
set_location_assignment PIN_207 -to c[9]
set_location_assignment PIN_216 -to c[10]
set_location_assignment PIN_218 -to c[11]
set_location_assignment PIN_220 -to c[12]
set_location_assignment PIN_222 -to c[13]
set_location_assignment PIN_224 -to c[14]
set_location_assignment PIN_230 -to c[15]
set_location_assignment PIN_181 -to r[0]
set_location_assignment PIN_183 -to r[1]
set_location_assignment PIN_185 -to r[2]
set_location_assignment PIN_187 -to r[3]
set_location_assignment PIN_189 -to r[4]
set_location_assignment PIN_195 -to r[5]
set_location_assignment PIN_197 -to r[6]
set_location_assignment PIN_199 -to r[7]
set_location_assignment PIN_201 -to r[8]
set_location_assignment PIN_203 -to r[9]
set_location_assignment PIN_214 -to r[10]
set_location_assignment PIN_217 -to r[11]
set_location_assignment PIN_219 -to r[12]
```

```
set_location_assignment PIN_221 -to r[13]
set_location_assignment PIN_223 -to r[14]
set_location_assignment PIN_226 -to r[15]
set_location_assignment PIN_230 -to c[15]
```

实验十三 MOORE机

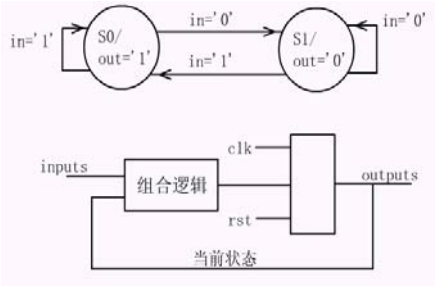
一、实验目的

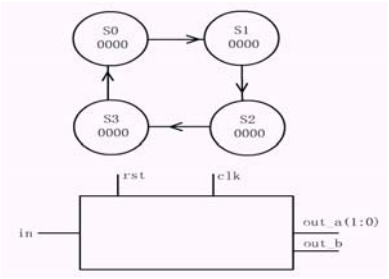
1、如图所示的状态图和状态机框图，设计并实现一个 output=state 类型的状态机，写出其 VHDL 源代码（包括 entity 和 architecture），当信号 RST='0'时，状态机应回到初始状态 S0。要求调试通过。

2、对于图中所示的状态图和状态机框图，输出信号是否存在"毛刺"没有要求，写出其 VHDL 源代码（包括 entity 和 architecture）并画出结果电路图，要求调试通过。

3、对于图中所示的状态图和状态机框图，写出其 VHDL 源代码（包括 entity 和 architectur 并画出结果电路图，要求调试通过。要求输出信号没有"毛刺"。

4、实验内容



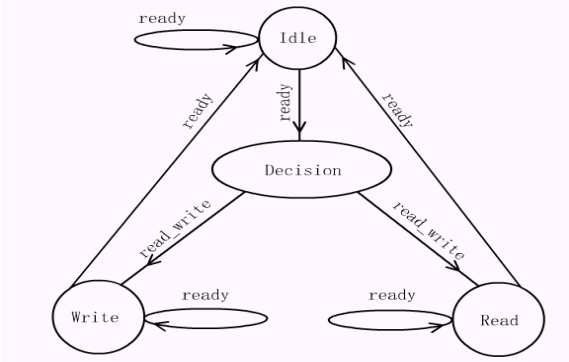


下面我们以一个存储控制器状态机的设计过程来介绍 Moore 机的设计。

设计要求：设计一个存储控制器状态机。能够根据微处理器的读写周期，分别对存储器输出写使能 WE 和读使能 OE 信号。

工作过程：存储控制器的输入信号为微处理器的就绪 READY 及读写 read_write 信号。当上电复位后，或 read 有效时，存储控制器开始工作，并在下一个时钟周期判断本次作业任务是读存储器还是写存储器。判断的依据是，当 read_write 有效时为读操作，否则为写操作。也就是说非读即写。读操作时，OE 信号有效，写操作时，WE 信号有效。当 READY 信号有效时，表示读本次作业处理完成，并使控制器恢复到初始状态。

控制器真值表和状态图如下：



存储控制器真值表

状态	输出	
	OE	WE
空闲（IDLE）	0	0
判断（DECISION）	0	0
写（WRITE）	0	1
读（READ）	1	0

程序原代码：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY moore IS
PORT(clk,ready,read_write: IN Std_Logic;
     oe,we                 : OUT Std_Logic);
END moore;
ARCHITECTURE state_machine OF Moore IS
TYPE state_type IS (idle,decision,read1,write1);
SIGNAL present_state,next_state: state_type;
BEGIN
  state_comb: PROCESS(present_state,ready,read_write)
  BEGIN
    CASE present_state IS
      WHEN idle =>
        oe<='0';
        we<='0';
        IF(ready='1') THEN
          next_state<=decision;
        ELSE
          next_state<=idle;
        END IF;
      WHEN decision =>
        oe<='0';
        we<='0';
        IF(read_write='1') THEN
          next_state<=read1;
        ELSE
```

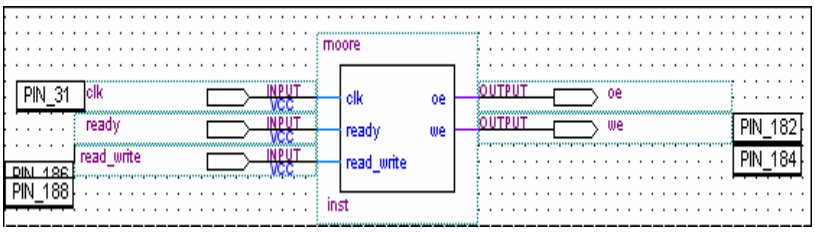
```
                next_state<=write1;
            END IF;
        WHEN read1 =>
            oe<='1';
            we<='0';
            IF(ready='1') THEN
                next_state<=idle;
            ELSE
                next_state<=read1;
            END IF;
        WHEN write1 =>
            oe<='0';
            we<='1';
            IF(ready='1') THEN
                next_state<=idle;
            ELSE
                next_state<=write1;
            END IF;
    END CASE;
END PROCESS state_comb;

state_clocked:PROCESS(clk)
BEGIN
    IF(rising_edge(clk)) THEN
        present_state<=next_state;
    END IF;
END PROCESS state_clocked;
END state_machine;
```

参考以上实例完成实验目的中所要求的 3 个状态机的设计。

2、 实验连线

同前，输入时钟信号接时钟电路的相应输出（CLK0~CLK5），复位信号接拨码开关或按键，输出信号接发光二极管。



管脚分配：

set_location_assignment PIN_31 -to clk

set_location_assignment PIN_182 -to oe

set_location_assignment PIN_184 -to we

set_location_assignment PIN_186 -to ready

set_location_assignment PIN_188 -to read_write

实验十四 MEALY机

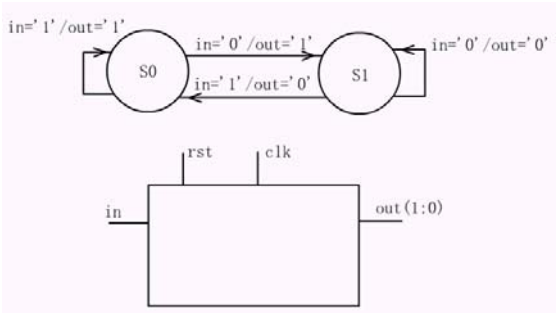
一、实验目的

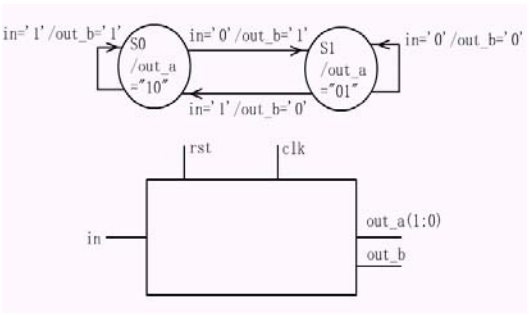
1、对于下图所示的状态图和状态机框图，将其实现为 Mealy 型状态机，输出信号是否存在“毛刺”没有要求，写出其 VHDL 源代码（包括 entity 和 architecture）并画出结果电路图，要求调试通过。

2、 重新完成第 1 个内容，要求输出信号没有“毛刺”。

3、对于图中所示的状态图和状态机框图，写出其 VHDL 源代码（包括 entity 和 architecture）并画出结果电路图，要求调试通过。要求输出信号没有“毛刺”。

二、实验内容

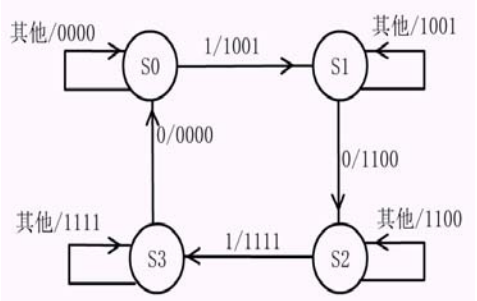


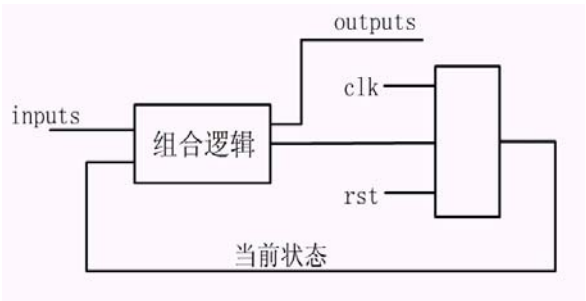


Mealy 型状态机和其等价的 Moore 型状态机相比，其输出变化要领先一个时钟周期。下面我们以一个实例来介绍 Mealy 机的设计过程。状态图和状态框图分别如下。

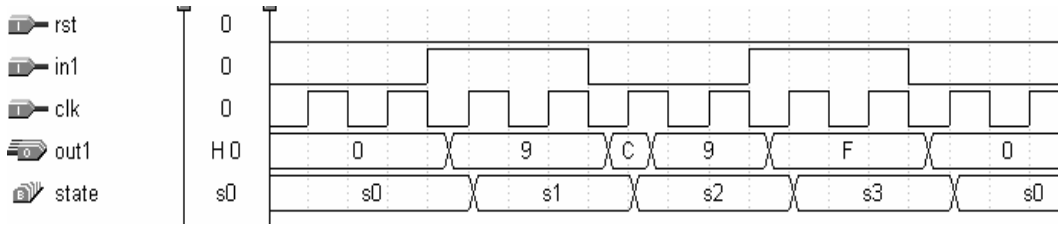
如下图中所示，Mealy 机的输出既和当前状态有关，又和所有输入信号有关。也就是说，一旦输入信号发生变化或状态发生变化，输出信号立即发生变化，因此在状态图中，一般把输出信号值画在状态变迁处。例如：假定当前状态为 S0，当输入信号为‘1’时，输出信号为“1001”；当输入信号不是‘1’时，输出信号为“0000”。

构造 Mealy 机的方法和 Moore 机相同，唯一的区别是：组合进程中的输出信号是当前状态和当前输入的函数。





输入输出信号仿真波形如下图 14-5：



例程如下：

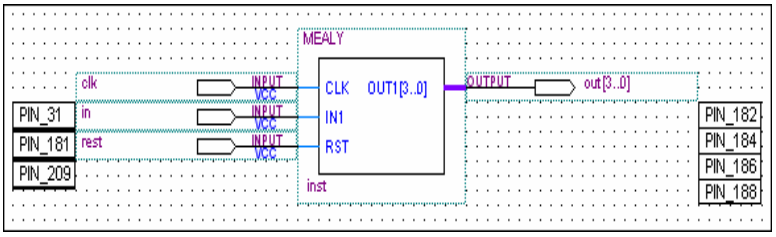
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MEALY IS
    PORT( CLK,IN1,RST:   IN    STD_LOGIC;
          OUT1:    OUT       STD_LOGIC_VECTOR(3 DOWNTO 0));
END;
ARCHITECTURE A OF MEALY IS
TYPE STATE_TYPE IS (S0,S1,S2,S3);
SIGNAL STATE:STATE_TYPE;
BEGIN
    MEALY_PROCESS:PROCESS(CLK,RST)
    BEGIN
        IF RST='1' THEN
            STATE<=S0;
        ELSIF RISING_EDGE(CLK) THEN
            CASE STATE IS
                WHEN S0 =>
                    IF IN1='1' THEN
                        STATE<=S1;
                    END IF;
                WHEN S1 =>
                    IF IN1='0' THEN
                        STATE<=S2;
                    END IF;
                WHEN S2 =>
                    IF IN1='1' THEN
                        STATE<=S3;
                    END IF;
                WHEN S3 =>
                    IF IN1='0' THEN
                        STATE<=S0;
                    END IF;
```

```
                END CASE;
            END IF;
        END PROCESS MEALY_PROCESS;
        OUTPUT_PROCESS:PROCESS(STATE,IN1)
        BEGIN
            CASE STATE IS
                WHEN S0 =>
                    IF IN1='1' THEN
                        OUT1<="1001";
                    ELSE
                        OUT1<="0000";
                    END IF;
                WHEN S1 =>
                    IF IN1='0' THEN
                        OUT1<="1100";
                    ELSE
                        OUT1<="1001";
                    END IF;
                WHEN S2 =>
                    IF IN1='1' THEN
                        OUT1<="1111";
                    ELSE
                        OUT1<="1001";
                    END IF;
                WHEN S3 =>
                    IF IN1='0' THEN
                        OUT1<="0000";
                    ELSE
                        OUT1<="1111";
                    END IF;
            END CASE;
        END PROCESS OUTPUT_PROCESS;
    END A;
```

参考以上实例完成实验目的中所要求的 3 个状态机的设计。

三、实验连线

同前，输入时钟信号接时钟电路的相应输出（CLK0~CLK5），复位信号接拨码开关或按键，输出信号接发光二极管。



管脚分配：

set_location_assignment PIN_31 -to clk

set_location_assignment PIN_181 -to in

set_location_assignment PIN_209 -to rest

set_location_assignment PIN_182 -to out[0]

set_location_assignment PIN_184 -to out[1]

set_location_assignment PIN_186 -to out[2]

set_location_assignment PIN_188 -to out[3]

实验十五 D/A接口

一、实验目的

1、学习 D/A 转换的原理；

2、熟悉 LPM_ROM 的建立方法；

3、学习正弦波发生器的设计；

4、学习使用 Signal Tap II 逻辑分析仪观察信号。

二、实验原理

首先参考实验十二的步骤，每一个时钟周期产生一个地址从 ROM 表中读取波形数据，送入 DA 转换器。使用 Signal Tap II 逻辑分析仪观察送出的波形。

三、实验步骤

1、建立工程命名为 sin，参考实验十二的步骤建立 64 个 8 位精度的正弦波数据表 sin.mif,接着建立 LPM_ROM(参考实验 12)命名为 DATA_ROM（也可以自己使用 C 语言按照 MIF 文件格式生成带正弦数据的 mif 文件），使用 sin.mif 将其初始化。

2、新建 vhdl 文件，保存为 Sin_VHDL.vhdl 用于地址发生器，生成顶层原理图，读取 ROM 中的数据。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
 --正弦波发生器文件
ENTITY   Sin_VHDL   IS
          PORT( RESET : IN STD_LOGIC;
               CLK     : IN STD_LOGIC;                    -- the inclock
               Addr_OUT: OUT STD_LOGIC_VECTOR(5 downto 0));--8bit sin data out
  END Sin_VHDL;
ARCHITECTURE DAC OF Sin_VHDL IS
 SIGNAL Q1 : STD_LOGIC_VECTOR (5 DOWNTO 0):="000000";--设定内部节点地址
 BEGIN
 PROCESS(CLK,RESET)
 BEGIN
      IF RESET='1' THEN
           IF CLK'EVENT AND CLK= '1' THEN
                Q1<=Q1+1;
           END IF;
      END IF;
```

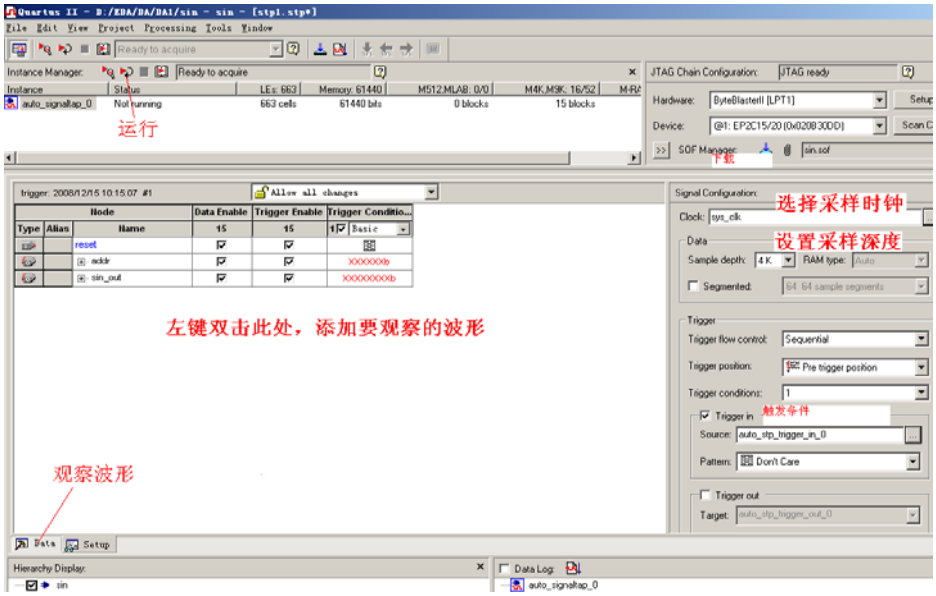
```
END PROCESS;
Addr_OUT<=Q1;
END DAC;
```

3、然后新建一个 Block Diagramm/Schematic File 原理图文件，在空白处双击添加生成的顶层原理图，并连接 input、output（输入输出管脚）重命名后保存文件 sin.bdf。

4、分配管脚，并建立 Signal Tap II 逻辑分析文件

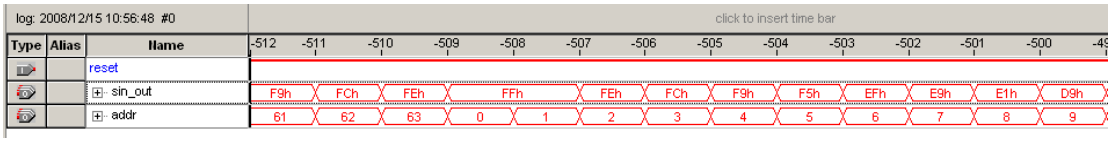
半编译后，Assignments →Pins 在 Location 中选择要分配的管脚，完毕。选择 File→New→Verification/Debugging Files→SignalTap II Logic Analyzer File，弹出对话框如下图所示，按图中设置，保存。

5、全编译完毕后，把 sof 文件通过 JTAG 下载到 FPGA，点击自动运行按钮观察波形数据。

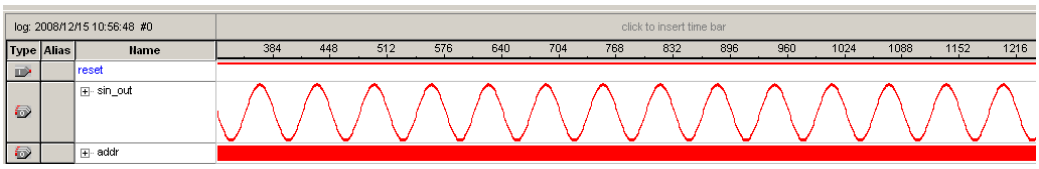


6、观察结果

在结果的数据区域单击鼠标左键或右键分别为扩展或压缩波形数据。数据展开口如下图所示。



选中观察的波形名称点击右键设置波形的显示方式，比如在 Sin_OUT 信号名上单击右键菜单选择 Bus Display Format→"Usigned Line Chart"观察。在波形上连续点击右键压缩波形后如下图所示



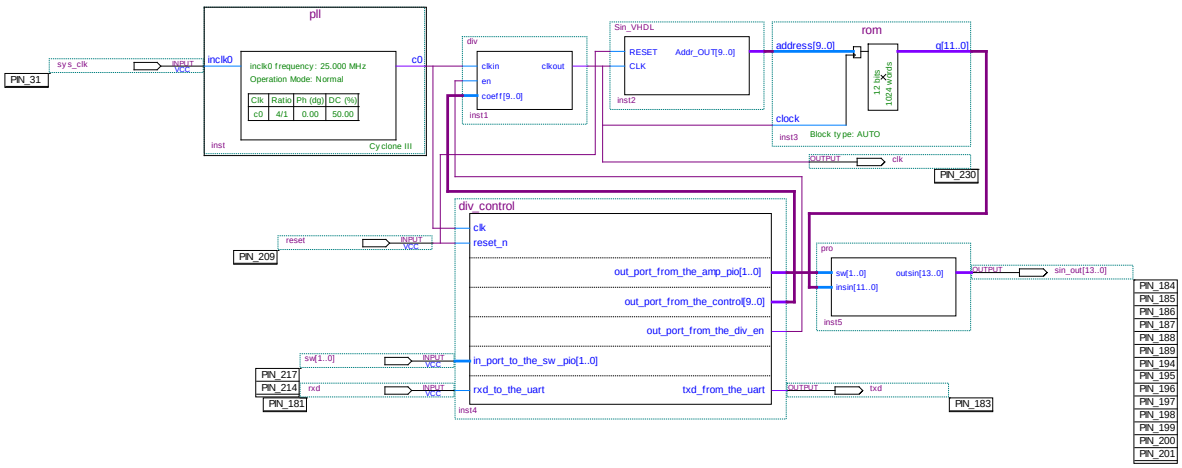
7、连线。Sin_OUT 的管脚连接 DA 模块（ADV7120）8 位数据，clk 连接 DA 模块的 clk 引脚

8、实验记录

1)、使用示波器观察 DA 的输出波形，并与逻辑分析仪波形做对比。

2)、已知 sys_clk 系统时钟为 25M，那么计算产生的该波形频率是多少，并与实际的频率比较 。

3)、使用 VHDL 描述一个分频器模块，将 sys_clk 分频输出，实现频率在一定范围内可调的正弦波输出。



管脚分配：

set_location_assignment PIN_31 -to sys_clk

set_location_assignment PIN_209 -to reset

set_location_assignment PIN_181 -to rxd

set_location_assignment PIN_183 -to txd

set_location_assignment PIN_185 -to clk

set_location_assignment PIN_182 -to sin_out[0]

set_location_assignment PIN_184 -to sin_out[1]

set_location_assignment PIN_186 -to sin_out[2]

set_location_assignment PIN_188 -to sin_out[3]

set_location_assignment PIN_194 -to sin_out[4]

set_location_assignment PIN_196 -to sin_out[5]

set_location_assignment PIN_198 -to sin_out[6]

set_location_assignment PIN_200 -to sin_out[7]

实验十六　A/D接口

一、实验目的

1、学习 A/D 转换的原理；

2、学习利用 FPGA 控制 A/D 转换。

3、学习 FPGA 内部锁相环 PLL 的使用

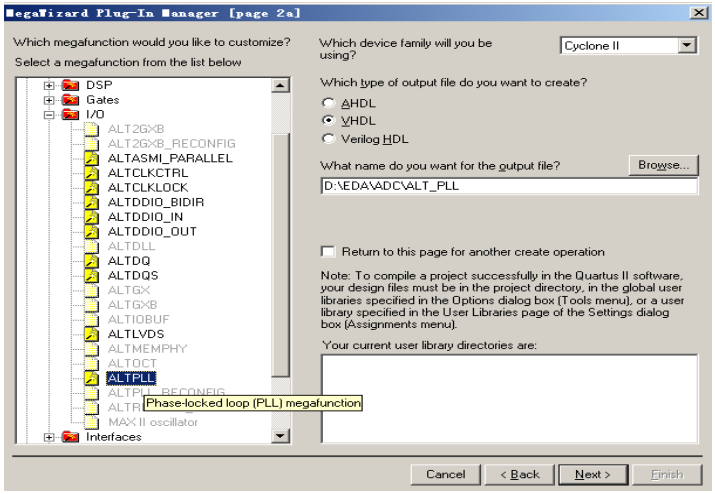
4、进一步熟悉嵌入式逻辑分析仪的使用方法。

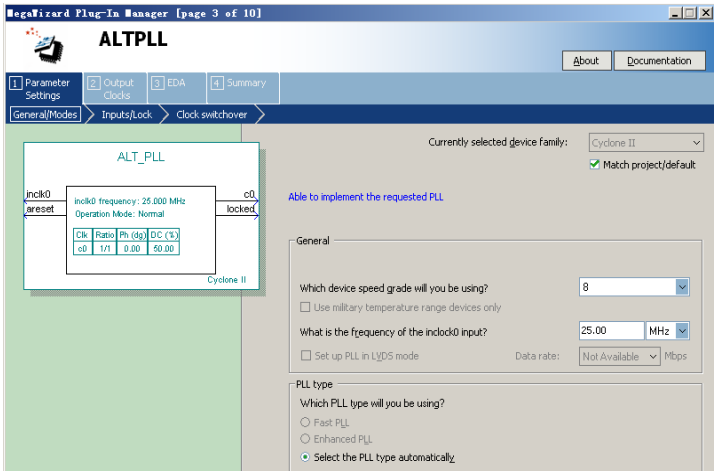
二、实验原理

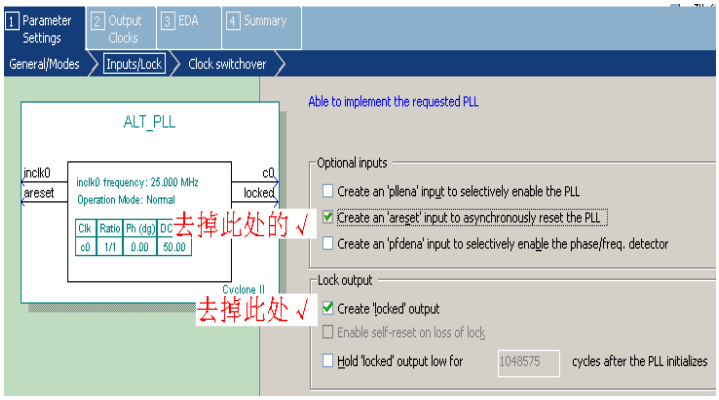
使用信号发生器产生峰峰值幅度不大于 1V 的高频（>1M）正弦波信号，加到 A/D 转换模块 AD9288 的输入端，利用嵌入式逻辑分析仪来观察波形。锁相环用来倍频产生 CLK100M 时钟控制 A/D 的转换速度。

三、实验步骤

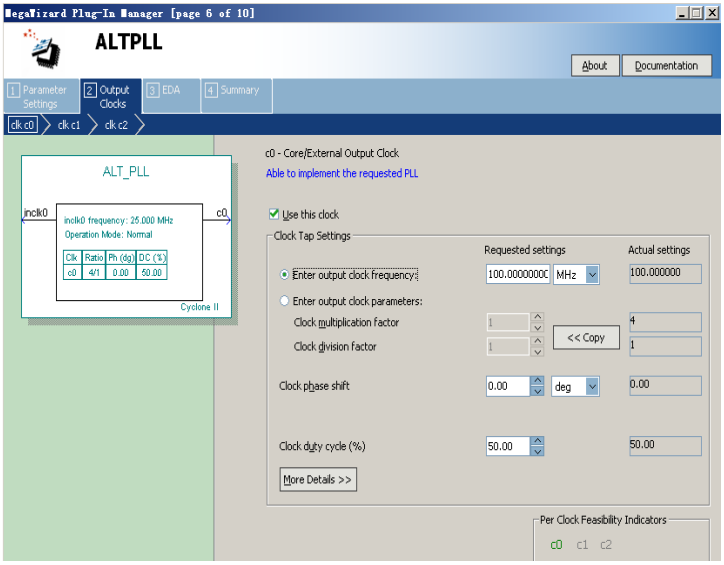
1、建立工程命名为 ADC，新建原理图文件保存为 ADC.bdf。然后选择 Tools→Megawizard Plug-In Manager 对话框如下图所示。点击 next，对话框中选择 I/O→ALTPLL,在右侧的输出文件路径中命名 PLL 为 ALT_PLL，点击 next，设置系统的输入时钟，本实验板的输入时钟固定为 25M。







修改后继续，去掉锁定的“√”后就剩下一个输入和一个输出管脚,连续点击 NEXT 进入下图所示的设置输出后的输出频率，本次设置为 100M，点击 NEXT 至结束。



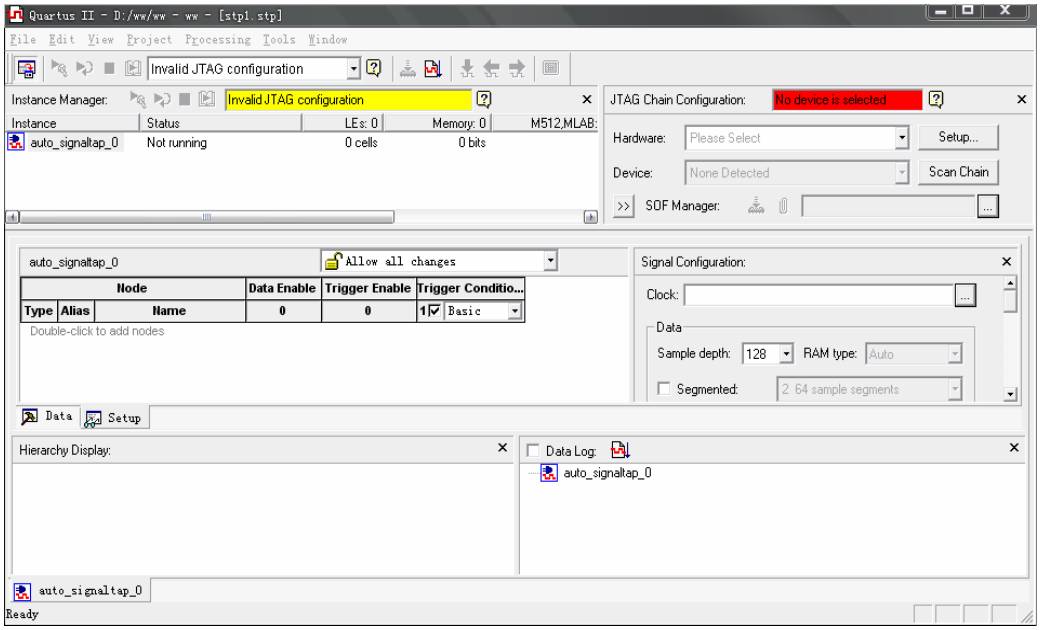
2、新建 vhdl 文件，输入一下参考代码，并保存名为 AD.VHD，生成顶层原理图，连接后半编译并分配管脚。

参考代码：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY AD IS
   PORT( CLKIN   :IN   STD_LOGIC;--CLK2 锁 pin17（锁相环频率输入口）
         CLKOUT :   OUT STD_LOGIC; --CLKDA，为双 ADC 提供时钟。
         DATIN   :IN   STD_LOGIC_VECTOR(7 DOWNTO 0);--data of DAc
         DATOUT :out   STD_LOGIC_VECTOR(7 DOWNTO 0));--out for signal tap ii
         END AD;
ARCHITECTURE DACC OF AD IS
    BEGIN
```

```
A:PROCESS(CLKIN)
  BEGIN
      IF CLKIN'EVENT AND CLKIN='1' THEN
        DATOUT<=DATIN;
      END IF;
  END PROCESS;
CLKOUT<=CLKIN;
END DACC ;
```

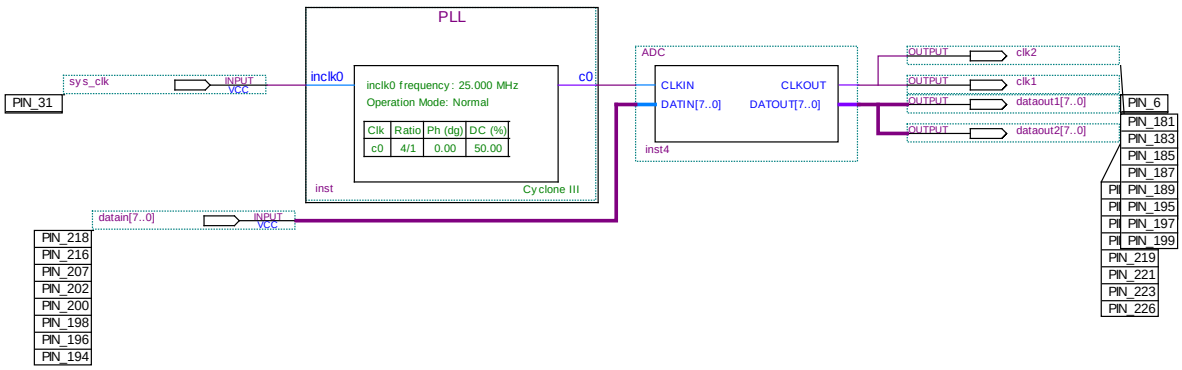
3、建立 Signal Tap II 逻辑分析文件。参考实验十五，设置如图所示，并保存文件。



4、全编译

点击主工具栏 全编译，完成后下载到 FPGA。

5、连接 datain(7..0)的引脚,到 AD 转换芯片 AD9288 的数据输出管脚，使用信号发生器产生 2M 的正弦波信号幅度设定为 500mv，接到 AD 转换的模拟输入端口，clk 信号的引脚接到 ad 的时钟输入脚。



6、实验记录

1)、在一定范围内改变信号发生器的输出波形的频率和幅度，运行 Signal Tap II 逻辑分析仪

观察信号波形变化情况。

2)、改变信号发生器的波形输出，使用三角波、方波，然后再观察逻辑分析仪的波形情况。

管脚分配：

```
set_location_assignment PIN_31 -to sys_clk
set_location_assignment PIN_209 -to reset
set_location_assignment PIN_218 -to datain[0]
set_location_assignment PIN_216 -to datain[1]
set_location_assignment PIN_207 -to datain[2]
set_location_assignment PIN_202 -to datain[3]
set_location_assignment PIN_200 -to datain[4]
set_location_assignment PIN_198 -to datain[5]
set_location_assignment PIN_196 -to datain[6]
set_location_assignment PIN_194 -to datain[7]
```

实验十七　动态数码管显示实验

一、实验目的

1、 学习数码管动态显示电路的连接方法；

2、 练习动态显示程序的结构及调试方法。

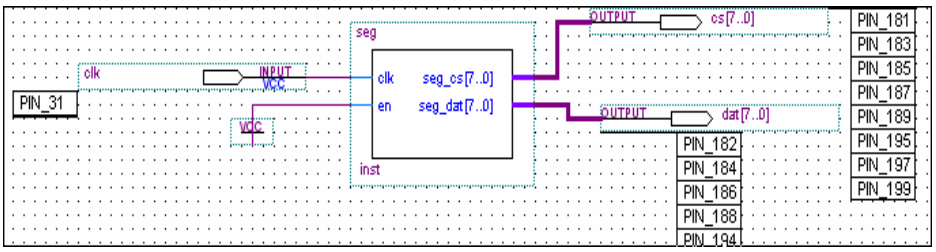
二、实验原理

动态显示是多个数码管交替显示，利用人的视觉暂留作用是人看到多个数码管同时显示。在编程时需要输出段选和位选信号，位选信号输出其中一个数码管，然后输出段码，使该数码管显示所需要的内容，延时一段时间后，再选中另一个数码管，再输出相应的段码，高速交替。

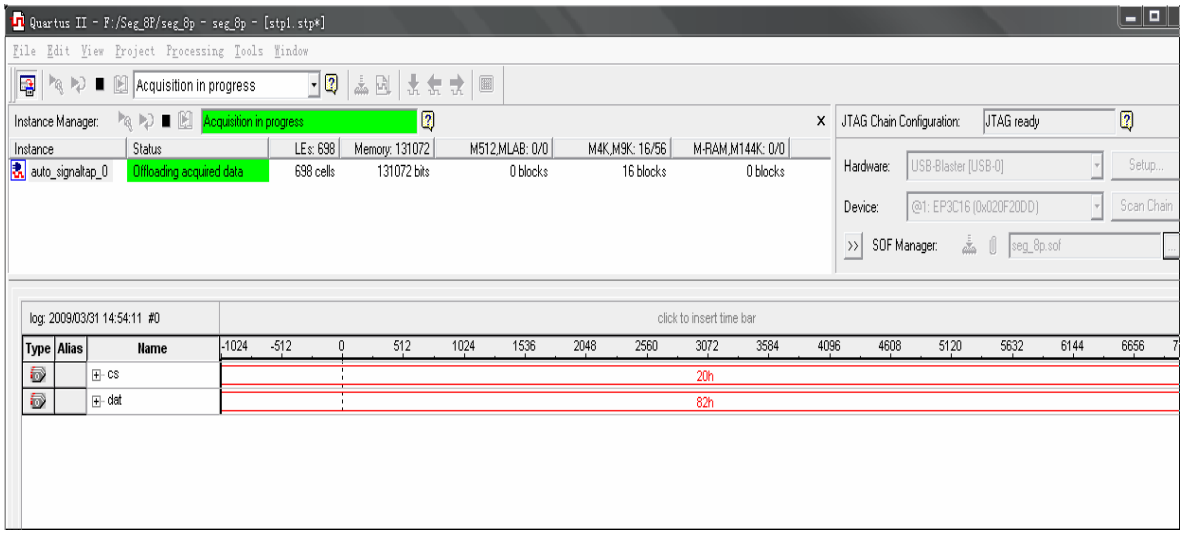
在动态显示中，各个位的延时时间长短是非常重要的，如果延时时间长，则会出现闪烁现象；如果延时时间短，则会出现显示暗且有重影。

三、实验步骤

1、 打开 quartus 环境，新建工程，命名为 seg_8p，并新建 VHDL 文件，编写显示程序，编译后创建模块图；新建 block 文件，加入模块后分配管脚。如下图所示：



2、新建 signalTAP 文件，设置后进行逻辑分析。



3、编译，下载，连线后观察数码管显示结果。

四、参考程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity seg is
```

```
        port (
                    signal clk        : IN    STD_LOGIC;
                    signal en         : IN STD_LOGIC;
                    signal seg_cs     : out STD_LOGIC_VECTOR (7 DOWNTO 0);
                    signal seg_dat : out STD_LOGIC_VECTOR (7 DOWNTO 0)
                );
end entity seg;
architecture behave of seg is
                    signal clk_ms :    STD_LOGIC;
                    shared variable clkcnt : integer:=0;
                    TYPE ram_array IS ARRAY (0 TO 9) OF STD_LOGIC_VECTOR(7 downto 0);
                    shared variable num : STD_LOGIC_VECTOR (3 DOWNTO 0);
                    shared variable cs : STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
                    shared variable lab : STD_LOGIC_VECTOR (8 DOWNTO 0):="000000000";
                    shared variable cs_num : STD_LOGIC_VECTOR (3 DOWNTO 0);
                    signal     select_seg: STD_LOGIC_VECTOR (7 DOWNTO 0):="10000000";
    begin
    A:process(clk)
      begin
        if(en='1') then
          if clk'event and clk='1' then
            if clkcnt<25000 then clkcnt:=clkcnt+1;
                clk_ms<='0';
            elsif clkcnt<50000 then clkcnt:=clkcnt+1;
                clk_ms<='1';
            else clkcnt:=0;
            end if;
          end if;
        end if;
      end process;
   B:PROCESS(clk,clk_ms)
     begin
    if(en='1') then
       if clk'event and clk='1' then
         case cs is
         when"0000"=>seg_cs<="00000001";
         when"0001"=>seg_cs<="00000010";
         when"0010"=>seg_cs<="00000100";
```

```
            when"0011"=>seg_cs<="00001000";
            when"0100"=>seg_cs<="00010000";
            when"0101"=>seg_cs<="00100000";
            when"0110"=>seg_cs<="01000000";
            when"0111"=>seg_cs<="10000000";
            when others=>seg_cs<="00000000";
            end case;
            case num is
            when"0000"=>seg_dat<="11000000";
            when"0001"=>seg_dat<="11111001";
            when"0010"=>seg_dat<="10100100";
            when"0011"=>seg_dat<="10110000";
            when"0100"=>seg_dat<="10011001";
            when"0101"=>seg_dat<="10010010";
            when"0110"=>seg_dat<="10000010";
            when"0111"=>seg_dat<="11111000";
            when"1000"=>seg_dat<="10000000";
            when"1001"=>seg_dat<="10010000";
            when others=>seg_dat<="11111111";
            end case;
        end if;
      end if;
    end process;
C: process(clk_ms)
    begin
    if(en='1') then
       if clk_ms'event and clk_ms='1' then
          num:=cs+cs_num;
          if(cs<=6) then cs:=cs+1;
          else cs:="0000";
          end if;
       end if;
    end if;
    end process;
D:   process(clk_ms)
    begin
    if(en='1') then
       if clk_ms'event and clk_ms='1' then
```

```
            if(lab<=300) then lab:=lab+1;
            else
                lab:="000000000";
                if(cs_num<9) then cs_num:=cs_num+1;
                else
                    cs_num:="0000";
                end if;
            end if;
          end if;
      end if;
      end process;
end architecture;
```

管脚分配：

```
      set_location_assignment PIN_31 -to clk
      set_location_assignment PIN_181 -to cs[7]
      set_location_assignment PIN_183 -to cs[6]
      set_location_assignment PIN_185 -to cs[5]
      set_location_assignment PIN_187 -to cs[4]
      set_location_assignment PIN_189 -to cs[3]
      set_location_assignment PIN_195 -to cs[2]
      set_location_assignment PIN_197 -to cs[1]
      set_location_assignment PIN_199 -to cs[0]
      set_location_assignment PIN_182 -to dat[7]
      set_location_assignment PIN_184 -to dat[6]
      set_location_assignment PIN_186 -to dat[5]
      set_location_assignment PIN_188 -to dat[4]
      set_location_assignment PIN_194 -to dat[3]
      set_location_assignment PIN_196 -to dat[2]
      set_location_assignment PIN_198 -to dat[1]
      set_location_assignment PIN_200 -to dat[0]
```

实验十八　频率信号源实验

一、实验目的

1、进一步熟悉 VHDL 语言字分频设计的应用；

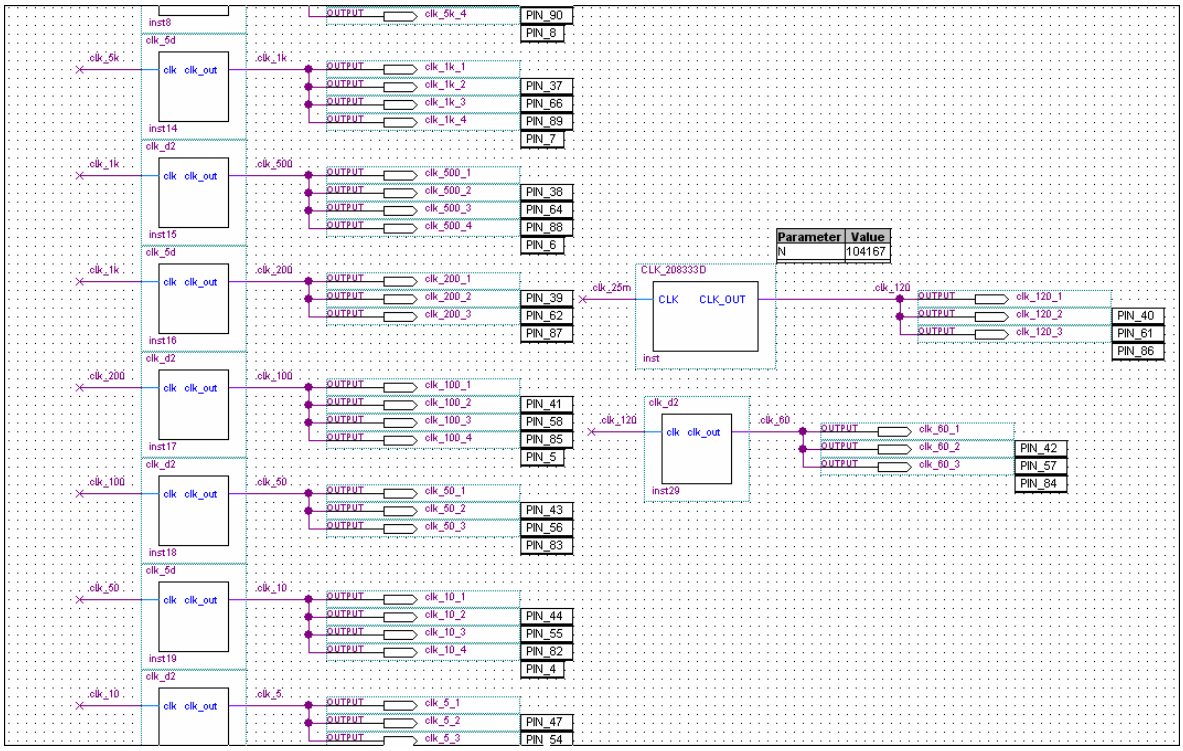
2、理解分频原理，学会各种形式的分频设计。

二、实验原理

分频器是数字电路中最常用的电路之一，在 FPGA 的设计中也是使用效率非常高的基本设计。基于 FPGA 实现的分频电路一般有两种方法：一是使用 FPGA 芯片内部提供的锁相环电路，如 ALTERA 提供的 PLL（Phase Locked Loop），Xilinx 提供的 DLL（Delay Locked Loop）；二是使用硬件描述语言，如 VHDL、Verilog HDL 等。使用锁相环电路有许多优点，如可以实现倍频；相位偏移；占空比可调等。但 FPGA 提供的锁相环个数极为有限，不能满足使用要求。因此使用硬件描述语言实现分频电路经常使用在数字电路设计中，消耗不多的逻辑单元就可以实现对时钟的操作，具有成本低、可编程等优点。本实验采用 MAX 系列 CPLD 来完成实验。

三、实验步骤

1、新建 Quartus 工程文件，命名为 signal_source，再新建 VHDL 文件，分别编写 5 分频器，2 分频器，4 分频器，104167 分频器等模块文件，编译后生成图形模块。新建 block 文件，加入生成的模块，分配管脚后编译，如下图所示。



2、 下载.sof 文件到 cpld 中，用示波器测量各个输出，观察频率是否正确。

四、参考程序

104167 分频模块参考程序：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY CLK_104167D IS
GENERIC(N:integer:=104167);
PORT(CLK:IN STD_LOGIC;
       CLK_OUT:BUFFER STD_LOGIC);
END ENTITY;
ARCHITECTURE behav OF CLK_104167D IS
SIGNAL COUNTER1,COUNTER2:integer:=0;
SIGNAL TEMP:STD_LOGIC;
BEGIN
PROCESS(CLK)
BEGIN
IF RISING_EDGE(CLK) THEN
 IF COUNTER1=N-1    THEN
     COUNTER1<=0;
 ELSE
     COUNTER1<=COUNTER1+1;
 END IF;
 IF COUNTER2=N-1    THEN
     COUNTER2<=0;
 ELSE
     COUNTER2<=COUNTER2+1;
 END IF;
END IF;
```

```
END PROCESS;
PROCESS(COUNTER1,COUNTER2)
BEGIN
  IF((N MOD 2)=1) THEN
    IF COUNTER1=1 THEN
      IF COUNTER2=0 THEN
        TEMP<='1';
      ELSE
        TEMP<='0';
      END IF;
    ELSIF COUNTER1=(N+1)/2 THEN
      IF COUNTER2=(N+1)/2 THEN
        TEMP<='1';
      ELSE
        TEMP<='0';
      END IF;
     ELSE
        TEMP<='0';
      END IF;
  ELSE
    IF COUNTER1=1 THEN
      TEMP<='1';
    ELSIF(COUNTER1=(N/2+1)) THEN
      TEMP<='1';
      ELSE
      TEMP<='0';
      END IF;
    END IF;
 END PROCESS;
 PROCESS(TEMP,CLK)
```

```
 BEGIN
 IF(N/=2)AND(N/=1) THEN
    IF TEMP'EVENT AND TEMP='1' THEN
       CLK_OUT<=NOT CLK_OUT;
    END IF;
 ELSIF(N=2) THEN
    IF(CLK'EVENT AND CLK='1') THEN
       CLK_OUT<=NOT CLK_OUT;
    END IF;
 ELSE
       CLK_OUT<=CLK;
    END IF;
END PROCESS;
END behav;
```

管脚分配：

```
    set_location_assignment PIN_12 -to clk_25m
      set_location_assignment PIN_1 -to clk_0_1_4
      set_location_assignment PIN_2 -to clk_0_5_4
      set_location_assignment PIN_3 -to clk_1_4
      set_location_assignment PIN_4 -to clk_10_4
      set_location_assignment PIN_5 -to clk_100_4
      set_location_assignment PIN_6 -to clk_500_4
      set_location_assignment PIN_7 -to clk_1k_4
      set_location_assignment PIN_8 -to clk_5k_4
      set_location_assignment PIN_14 -to clk_10k_4
      set_location_assignment PIN_15 -to clk_50k_4
      set_location_assignment PIN_16 -to clk_100k_4
      set_location_assignment PIN_17 -to clk_250k_4
      set_location_assignment PIN_18 -to clk_500k_4
```

```
set_location_assignment PIN_19 -to clk_1m_4
set_location_assignment PIN_20 -to clk_5m_4
set_location_assignment PIN_21 -to clk_25m_4
set_location_assignment PIN_26 -to clk_25m_1
set_location_assignment PIN_27 -to clk_5m_1
set_location_assignment PIN_28 -to clk_1m_1
set_location_assignment PIN_29 -to clk_500k_1
set_location_assignment PIN_30 -to clk_250k_1
set_location_assignment PIN_33 -to clk_100k_1
set_location_assignment PIN_34 -to clk_40k_1
set_location_assignment PIN_35 -to clk_10k_1
set_location_assignment PIN_36 -to clk_5k_1
set_location_assignment PIN_37 -to clk_1k_1
set_location_assignment PIN_38 -to clk_500_1
set_location_assignment PIN_39 -to clk_200_1
set_location_assignment PIN_40 -to clk_120_1
set_location_assignment PIN_41 -to clk_100_1
set_location_assignment PIN_42 -to clk_60_1
set_location_assignment PIN_43 -to clk_50_1
set_location_assignment PIN_44 -to clk_10_1
set_location_assignment PIN_47 -to clk_5_1
set_location_assignment PIN_48 -to clk_1_1
set_location_assignment PIN_49 -to clk_0_5_1
set_location_assignment PIN_50 -to clk_0_1_1
set_location_assignment PIN_51 -to clk_0_1_2
set_location_assignment PIN_52 -to clk_0_5_2
set_location_assignment PIN_53 -to clk_1_2
set_location_assignment PIN_54 -to clk_5_2
set_location_assignment PIN_55 -to clk_10_2
set_location_assignment PIN_56 -to clk_50_2
```

```
set_location_assignment PIN_57 -to clk_60_2
set_location_assignment PIN_58 -to clk_100_2
set_location_assignment PIN_61 -to clk_120_2
set_location_assignment PIN_62 -to clk_200_2
set_location_assignment PIN_64 -to clk_500_2
set_location_assignment PIN_66 -to clk_1k_2
set_location_assignment PIN_67 -to clk_5k_2
set_location_assignment PIN_68 -to clk_10k_2
set_location_assignment PIN_69 -to clk_40k_2
set_location_assignment PIN_70 -to clk_100k_2
set_location_assignment PIN_71 -to clk_250k_2
set_location_assignment PIN_72 -to clk_500k_2
set_location_assignment PIN_73 -to clk_1m_2
set_location_assignment PIN_74 -to clk_5m_2
set_location_assignment PIN_75 -to clk_25m_2
set_location_assignment PIN_76 -to clk_0_1_3
set_location_assignment PIN_77 -to clk_0_5_3
set_location_assignment PIN_78 -to clk_1_3
set_location_assignment PIN_81 -to clk_5_3
set_location_assignment PIN_82 -to clk_10_3
set_location_assignment PIN_83 -to clk_50_3
set_location_assignment PIN_84 -to clk_60_3
set_location_assignment PIN_85 -to clk_100_3
set_location_assignment PIN_86 -to clk_120_3
set_location_assignment PIN_87 -to clk_200_3
set_location_assignment PIN_88 -to clk_500_3
set_location_assignment PIN_89 -to clk_1k_3
set_location_assignment PIN_90 -to clk_5k_3
set_location_assignment PIN_91 -to clk_10k_3
set_location_assignment PIN_92 -to clk_40k_3
```

```
set_location_assignment PIN_95 -to clk_100k_3
set_location_assignment PIN_96 -to clk_250k_3
set_location_assignment PIN_97 -to clk_500k_3
set_location_assignment PIN_98 -to clk_1m_3
set_location_assignment PIN_99 -to clk_5m_3
set_location_assignment PIN_100 -to clk_25m_3
```

第四章 SOPC系统基本实验

实验一 Hello-Led流水灯实验

一、实验目的

熟悉 SOPC Builder 和 Nios-II IDE 的使用方法和步骤，了解 Nios-II 软核的配置方法，初步掌握对 IO 口的操作。

二、实验模块

FPGA 主控制板模块

三、实验原理

采用 NIOS 软核实现跑马灯循环闪烁。

四、实验内容

选择例程 Hello_led,实现 8 个 led 灯的循环亮灭；实验者也可以适当修改程序来实现其他效果。

五、实验步骤

1、利用 Quartus II 软件建立工程命名为 Hello_led,点击主工具栏上的 图标启动 SOPC Builder 如图 1.1 所示，建立 CPU 命名为 Helloled，如图 1.2 所示 。

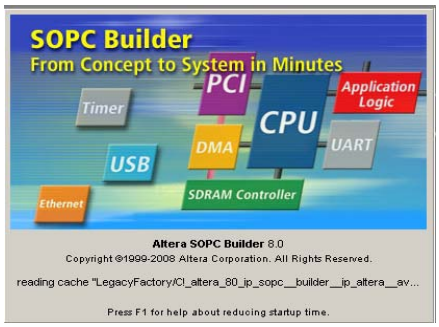


图 4.1.1 启动 SOPC Builder

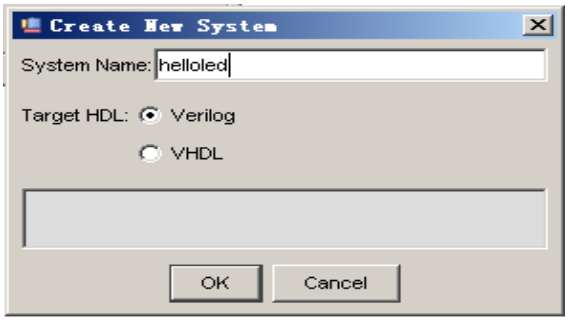


图 4.1.2 建立名为 helloled 的 CPU

2、点击 OK 后，打开界面如图 1.3 所示。修改系统时钟为 25M(视实际需要而定)

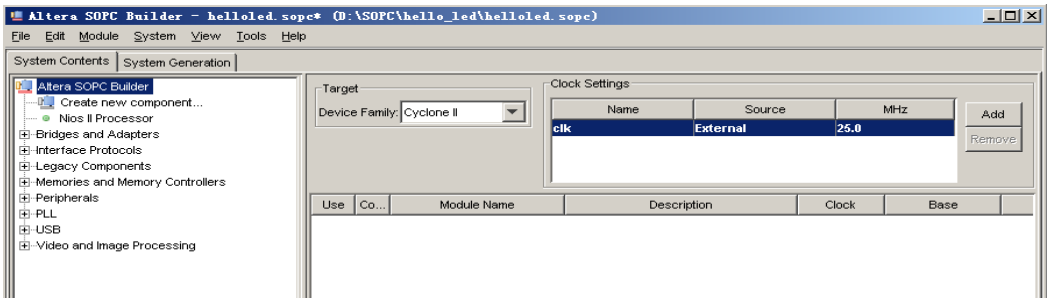


图 4.1.3 打开 SOPC Builder 构造环境

3、在图 1.3 左侧菜单中选择 Memeries and Memery Contorlers→ On-Chip Memery(RAM or ROM),设置如图 1.4 所示。建立 10k 大小的片上 RAM，点击 OK 后退出。

2、 添加 Nios II Procesor 处理器

在左侧菜单中选择 Nios II Procesor 双击，弹出对话框如图 1.5 所示，设置为”fast”快速处理器并设置其复位地址在刚建立的 onchip_mem 中。

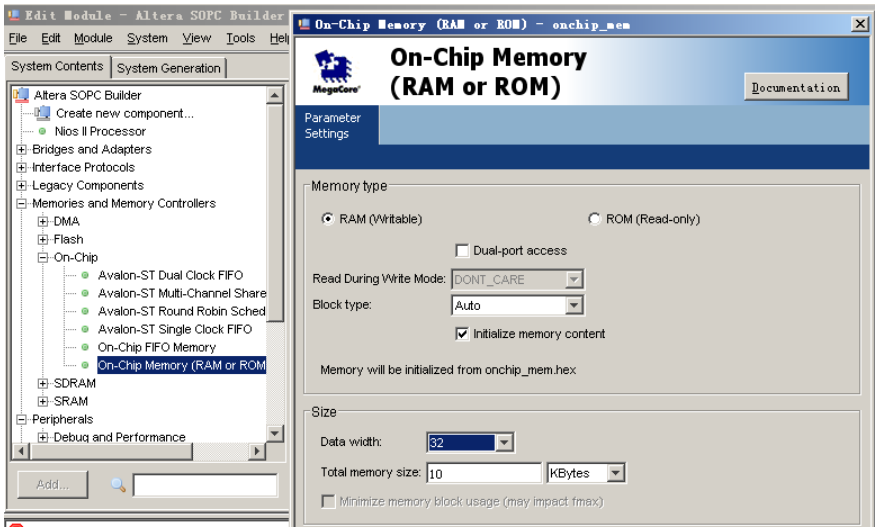


图 4.1.4 添加片上 RAM

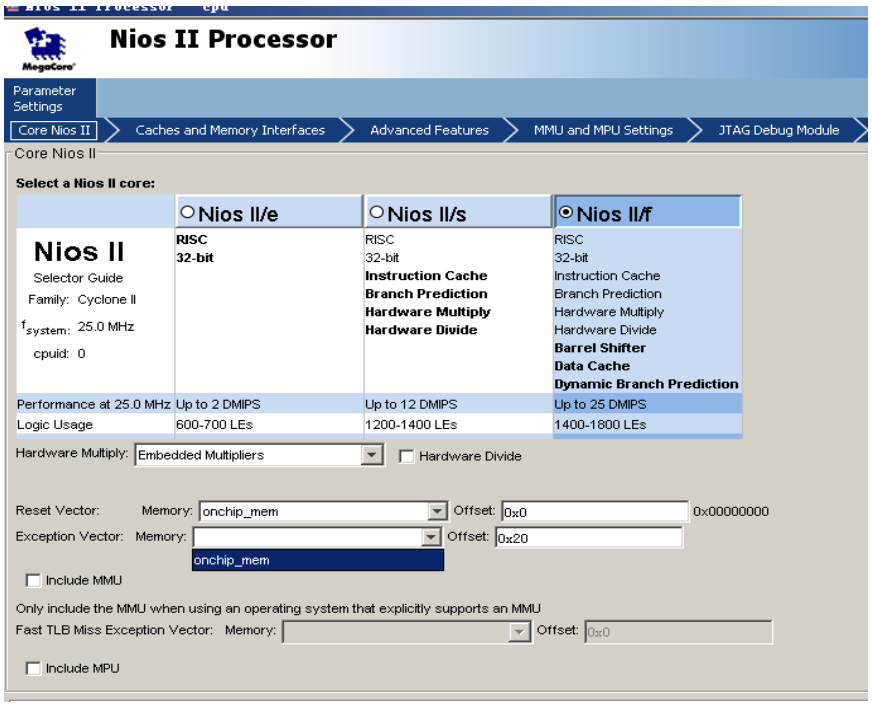


图 4.1.5 Nios II 处理器 RAM 设置

5、添加 PIO 模块

如图 1.6 所示，双击 PIO，弹出对话框设置为 8 路输出，并命名为 LED_PIO。

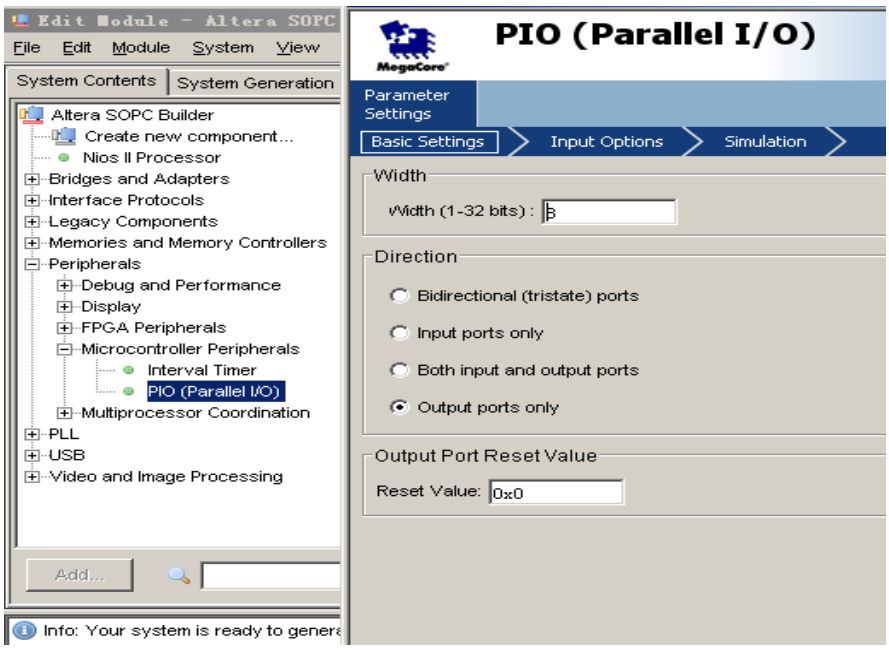


图 4.1.6 添加 PIO 模块

6、生成 CPU

选添加后的 pio 组件，然后按 F2 重命名为 LED_PIO,如图 1.7 所示。点击 Next，然后点击”Generate”开始生成 CPU，如图 1.8 所示。完成后点击 EXIT 退出。

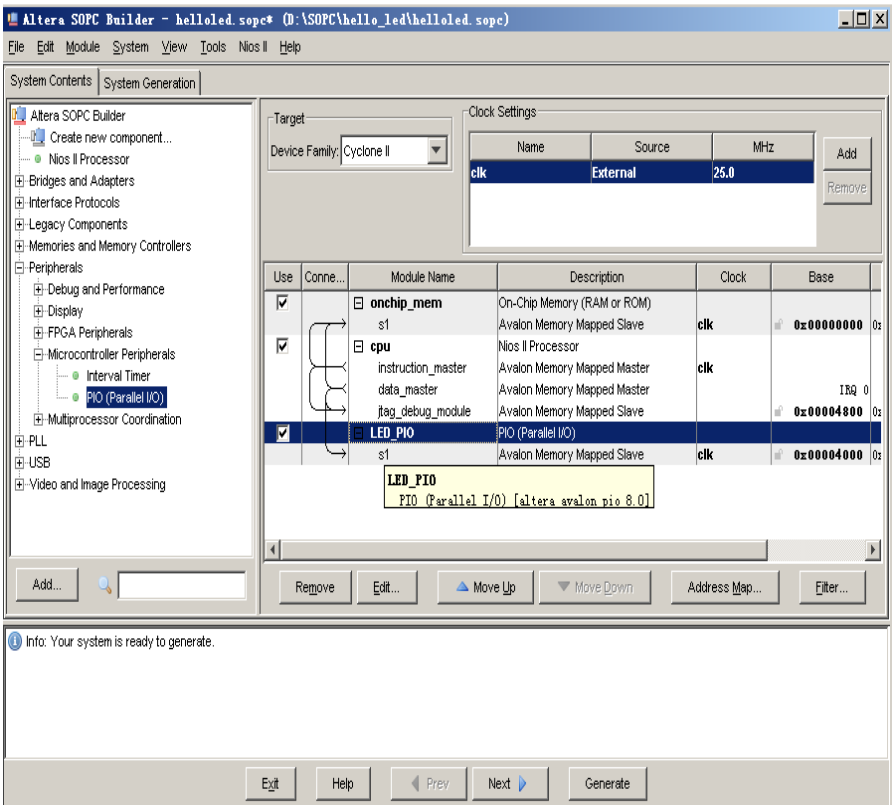


图 4.1.7 重命名完成

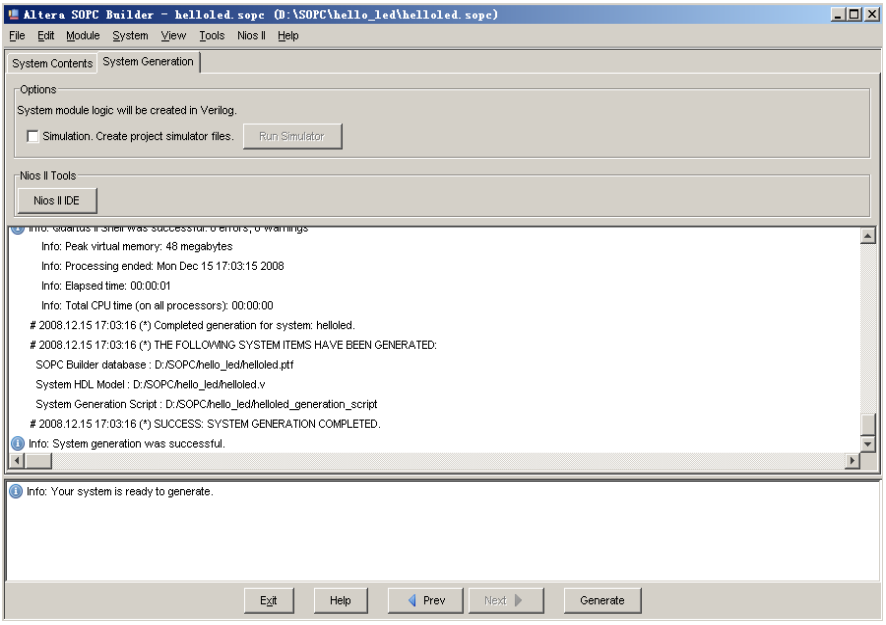


图 4.1.8　生成 CPU 成功

7、建立 File－New－Block Diagram/Schematic File,加入生成的 Helloled CPU 与输入输出 I/O 口，并在重命名后分配管脚，如图 1.9 所示，全编译后下载到 FPGA。注意**复位引脚必须拉高电平**。

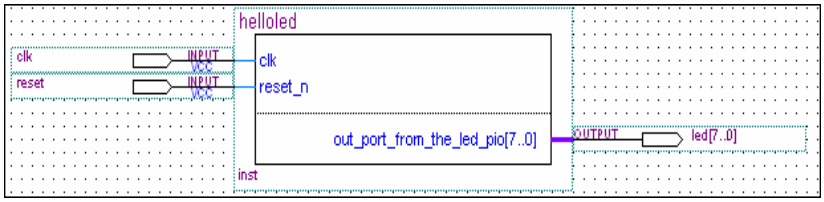


图 4.1.9 管脚分配后的原理图文件

8、启动 Nios II IDE 如图 1.10。

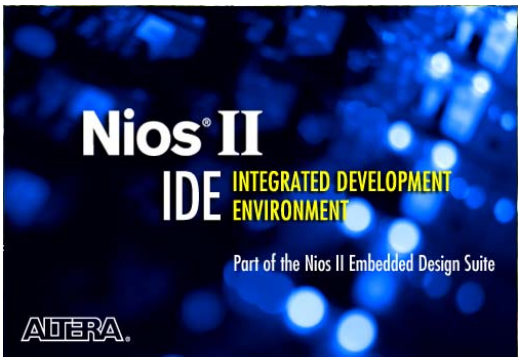


图 4.1.10 启动 IDE

图 4.1.11　进入环境

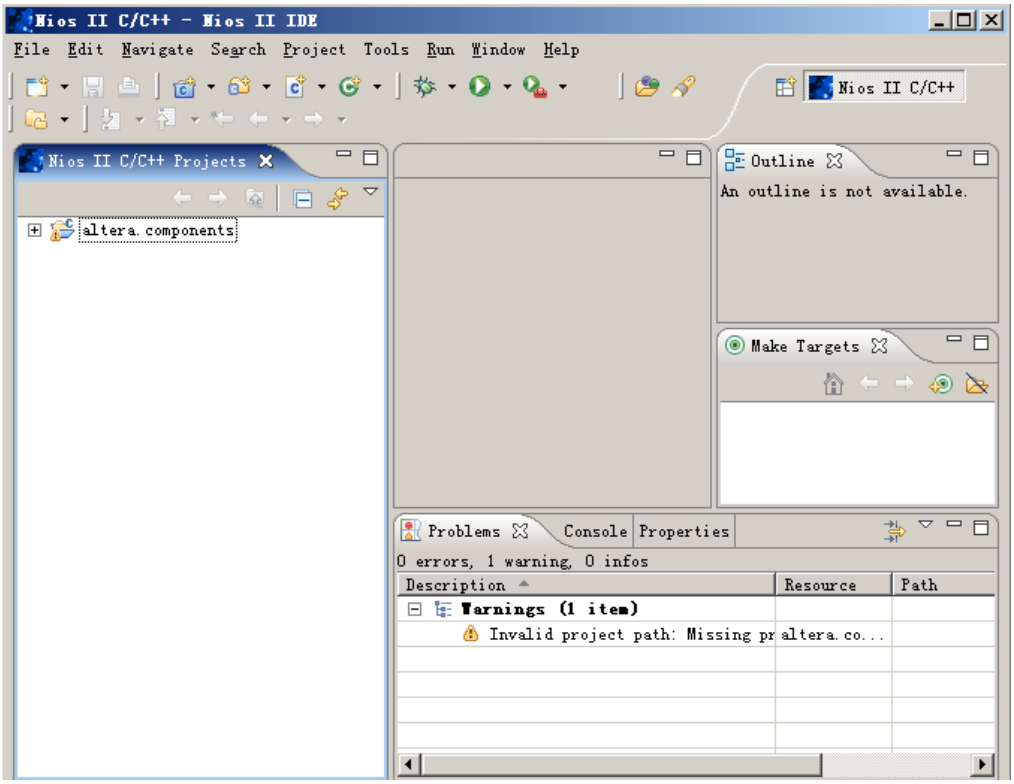


图 4.1.12 运行环境

9、选择工作目录

点击 File→Switch Workspace,弹出对话框如图 1.13 所示。注意路径不能有空格或汉字。路径必须为 Quatus II 所建工程的路径。

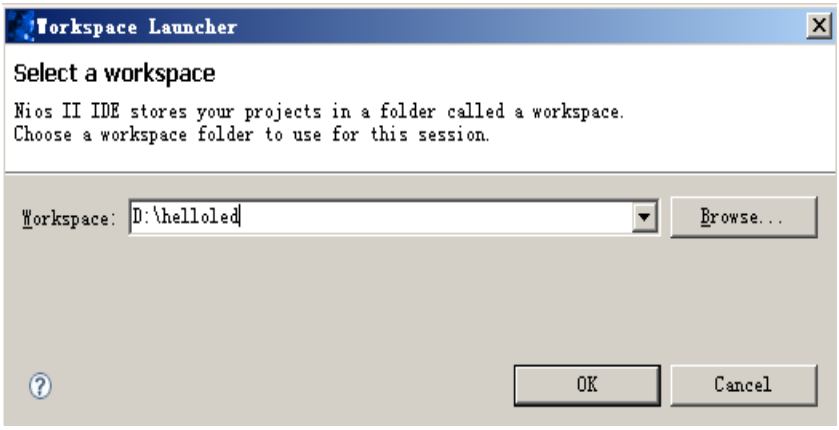


图 4.1.13 更换环境路径

重新设置工作目录路径，建立新的 C/C++应用程序；选择 helloled.ptf CPU 和空实例如图 1.14 所示，点击 NEXT，然后点击”Finish”完成。

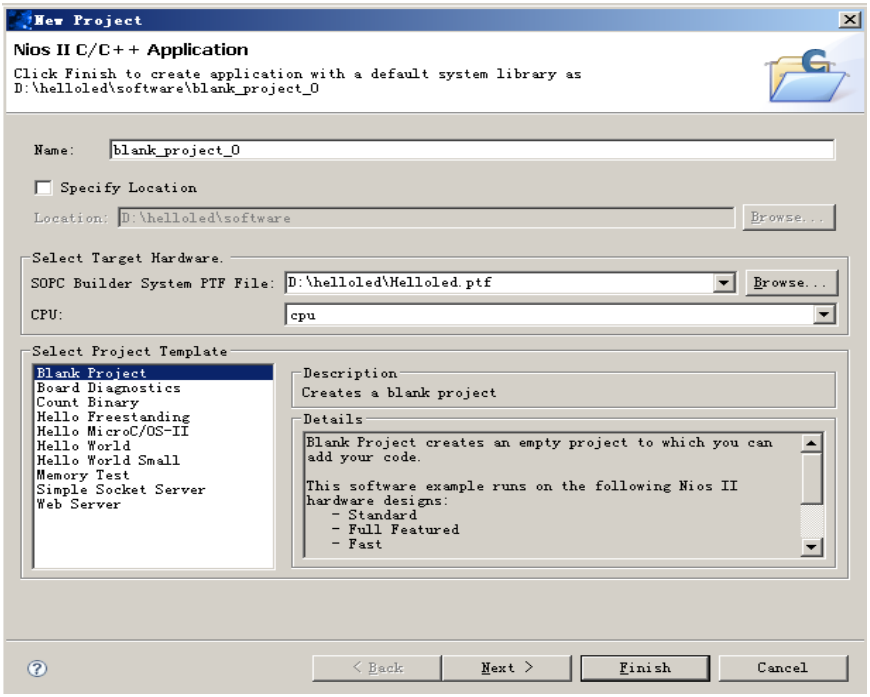


图 4.1.14　建立应用程序

10、新建 C 文件

选择 File→New→Source File,设置如图 1.15 所示。

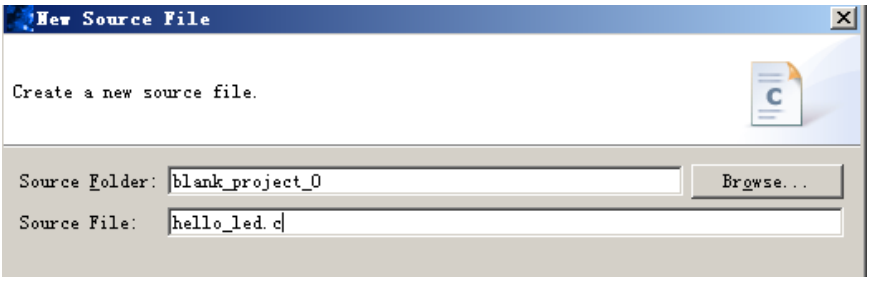


图 4.1.15　新建 c 文件

11、然后输入以下程序，并保存。

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
int main (void) __attribute__ ((weak, alias ("alt_main")));
int alt_main (void)
{   alt_u8 led = 0x2;
  alt_u8 dir = 0;
  volatile int i;
   while (1)
  {     if (led & 0x81)
    {          dir = (dir ^ 0x1);}
    if (dir)
```

```
    { led = led >> 1;
    } else
    {led = led << 1;
    }
    IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, led);
    i = 0;
    while (i<200000) i++;
    }
  return 0;
}
```

12、编译

在如图所示位置点击鼠标右键选择 Build Project ，开始编译工程。

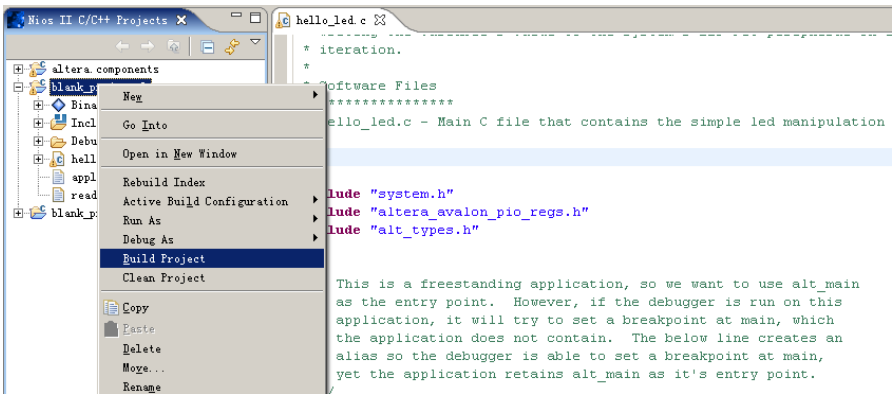

Nios II C/C++ Projects
hello_led.c
altera_components
New
Go Into
Open in New Window
Rebuild Index
Active Build Configuration
Run As
Debug As
Build Project
Clean Project
Copy
Paste
Delete
Move...
Rename
* iteration.
* Software Files
hello_led.c - Main C file that contains the simple led manipulation
lude "system.h"
lude "altera_avalon_pio_regs.h"
lude "alt_types.h"
This is a freestanding application, so we want to use alt_main
as the entry point. However, if the debugger is run on this
application, it will try to set a breakpoint at main, which
the application does not contain. The below line creates an
alias so the debugger is able to set a breakpoint at main,
yet the application retains alt_main as it's entry point.

图 4.1.16 编译工程

13、下载。首先启动 Quartus II 下的 Programer,把 Hello_led.sof 文件配置到 FPGA 中，LED_PIO 对应的引脚连接 8 个 led 发光二极管。

14、在 Nios II IDE 中图中位置单击右键 Run as →Nios IIHardware.如图 1.17， 开始运行程序，成功后如图 1.18 所示，可以观察到 8 个 LED 灯的循环亮灭。

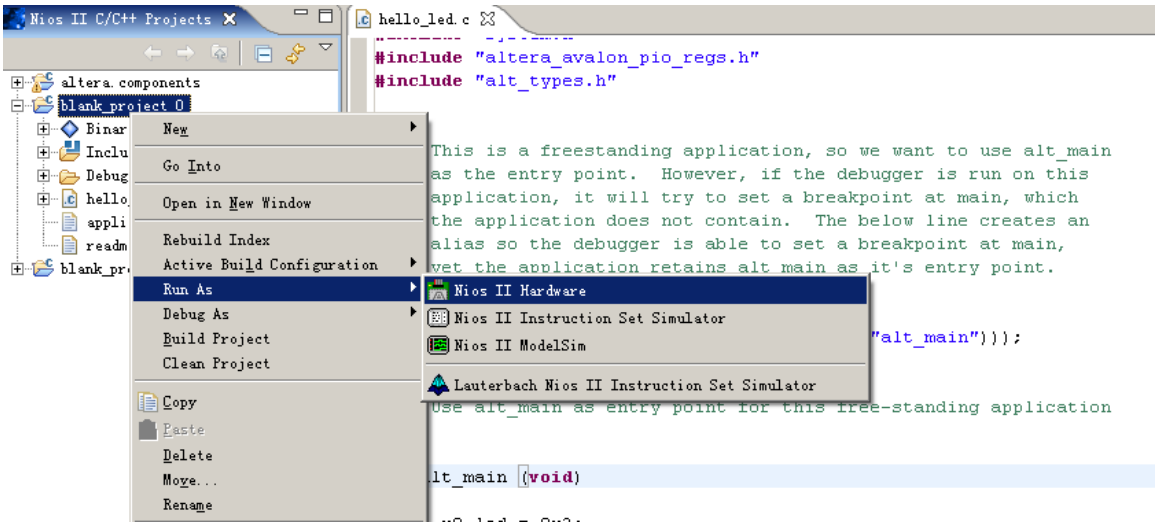


图 4.1.17 运行程序

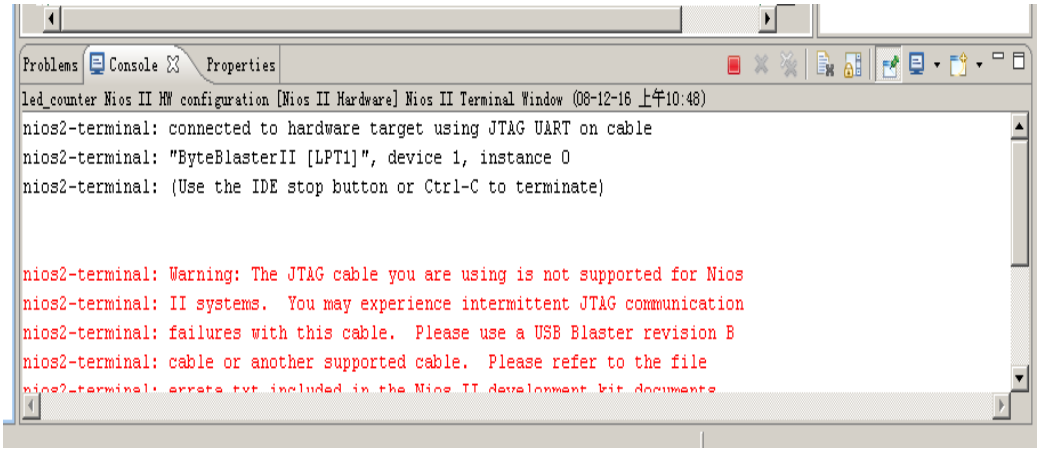

Problems
Console
Properties
led_counter Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (08-12-16 上午10:48)
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "ByteBlasterII [LPT1]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)
nios2-terminal: Warning: The JTAG cable you are using is not supported for Nios
nios2-terminal: II systems. You may experience intermittent JTAG communication
nios2-terminal: failures with this cable. Please use a USB Blaster revision B
nios2-terminal: cable or another supported cable. Please refer to the file

图 4.1.18 运行成功

15、实验记录

要求修改程序，实验流水灯的多个效果。

管脚分配：

set_location_assignment PIN_31 -to IN_CLK

set_location_assignment PIN_209 -to IN_RESET

set_location_assignment PIN_182 -to LED[0]

set_location_assignment PIN_184 -to LED[1]

set_location_assignment PIN_186 -to LED[2]

set_location_assignment PIN_188 -to LED[3]

set_location_assignment PIN_194 -to LED[4]

set_location_assignment PIN_196 -to LED[5]

set_location_assignment PIN_198 -to LED[6]

set_location_assignment PIN_200 -to LED[7]

实验二　数码管显示实验

一、实验目的

进一步熟悉 SOPC Builder 和 Nios-II IDE 的使用方法和步骤，掌握 Nios-II 软核的配置方法，掌握对 IO 口的写操作。

二、实验模块

FPGA 主控制板模块

三、实验原理

通过程序向数码管送相应的显示代码，即可点亮数码管段，实现数字显示。

四、实验内容

编写程序，实现两个数码管的 0—99 的循环计数；实验者也可以适当修改程序来实现其他效果。

五、实验步骤

1、利用 Quartus II 软件建立工程命名为 led_counter,启动 SOPC Builder 建立 CPU 命名为 counter；

2、为 CPU 添加 Nios II 软核，选择 Nios II/S,建立片内 2K 的 on-chip ram 命名为 RAM，并为 CPU 添加复位地址为 RAM 地址。添加 PIO 模块，设置为两个 8 路输出，并命名为 LED1、LED2; 然后生成 CPU；

3、建立顶层原理图文件加入生成的 counter CPU 与输入输出 I/O 口，并在重命名后分配管脚（系统时钟 pin31，复位 pin209），用排线按顺序连接硬件电路。编译后如图 1 所示：**复位可直接使用 vcc 拉高电平，也可使用管脚实现复位控制。**

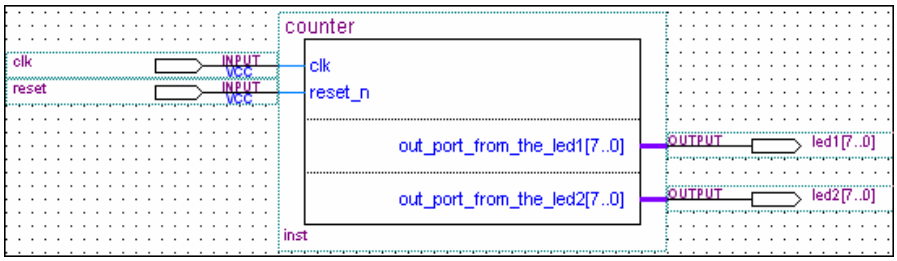


图 4.2.1 顶层原理图

4、 启动 Nios II IDE 并重新设置工作目录路径，建立新的 C/C++应用程序；选择 counter.ptf CPU 和实例如图 2 所示，选择空工程然后重命名工程为 led_counter，接着创建工程。

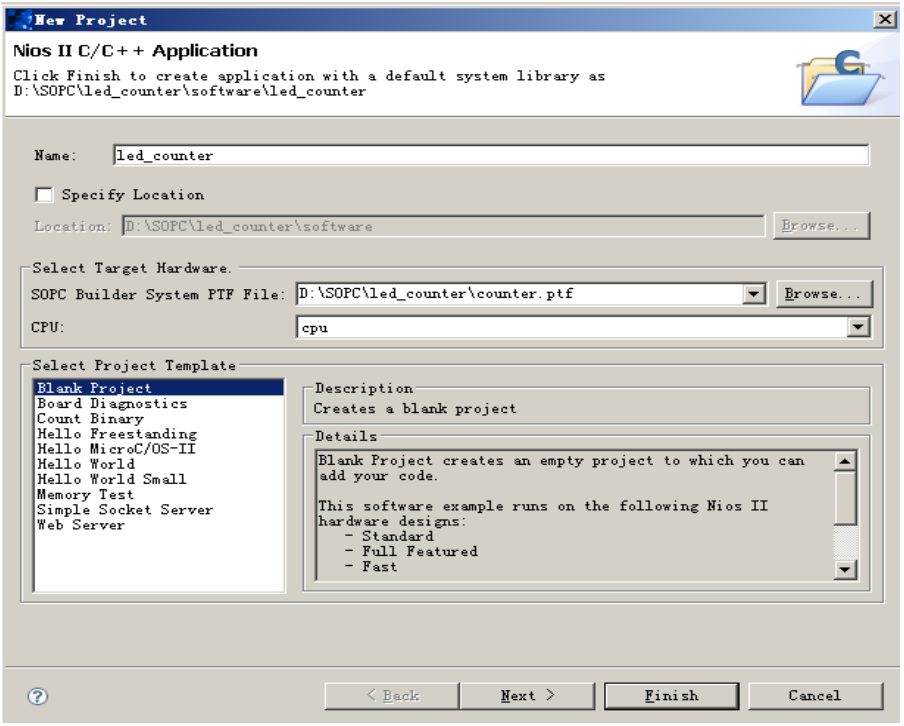


图 4.2.2 新建 NiosII 工程

5、新建 C 文件

如图 2.3 所示，首先选中左侧新建的工程，点击右键在弹出菜单中选择 New→Source File 新建 C 文件，然后重命名，如图 2.4 所示。完成再在其中输入参考代码。

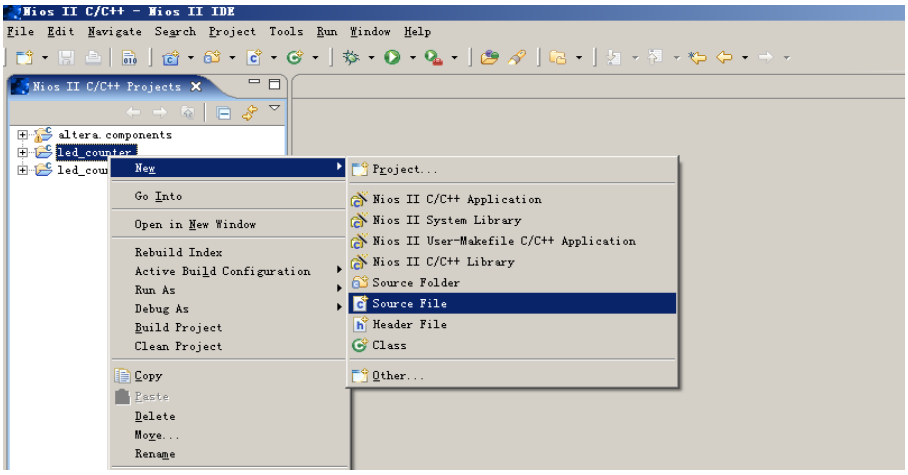


图 4.2.3 新建 C 文件

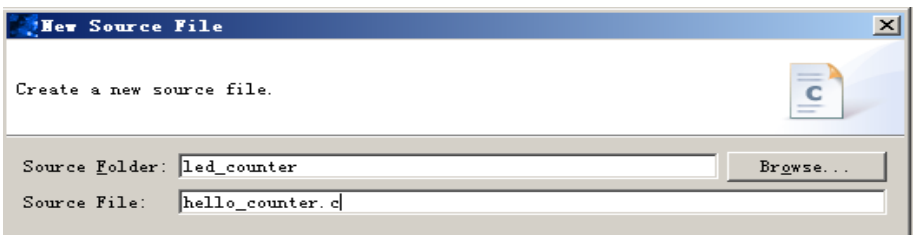


图 4.2.4 重命名 C 文件

6、编译文件

在图 2.3 中首先选中左侧新建的工程，点击右键在弹出菜单中选择 Bulid Project

7、下载。首先启动 Quartus II 下的 Programer,把 led_counter.sof 文件配置到 FPGA 中，按引脚分配连接两个数码管，然后在 Nios II IDE 的图 2.3 中选择 Run as→Nios II Hardware.运行 ch 成功后，可以观察到两个数码管的 0-99 的循环显示。

8、参考程序：

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
int main (void) __attribute__ ((weak, alias ("alt_main")));
int alt_main (void)
{volatile int i;
        alt_u8 counter;
        alt_u8 led_table[10]={0xC0,0XF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
       while(1)
       {if(counter<99)counter=counter+1;
        else counter=0;
        IOWR_ALTERA_AVALON_PIO_DATA(LED1_BASE,led_table[counter/10]);
       IOWR_ALTERA_AVALON_PIO_DATA(LED2_BASE,led_table[counter%10]);
       i = 0;
       while (i<300000)
       i++;
         }
       return 0;
 }
```

9、试验记录

要求修改程序实现 0-99 的递加后再递减的循环显示效果。

管脚分配：

set_location_assignment PIN_31 -to CLK

```
set_location_assignment PIN_182 -to LED1[0]
set_location_assignment PIN_184 -to LED1[1]
set_location_assignment PIN_186 -to LED1[2]
set_location_assignment PIN_188 -to LED1[3]
set_location_assignment PIN_194 -to LED1[4]
set_location_assignment PIN_196 -to LED1[5]
set_location_assignment PIN_198 -to LED1[6]
set_location_assignment PIN_200 -to LED1[7]
set_location_assignment PIN_202 -to LED2[0]
set_location_assignment PIN_207 -to LED2[1]
set_location_assignment PIN_216 -to LED2[2]
set_location_assignment PIN_218 -to LED2[3]
set_location_assignment PIN_220 -to LED2[4]
set_location_assignment PIN_222 -to LED2[5]
set_location_assignment PIN_224 -to LED2[6]
set_location_assignment PIN_230 -to LED2[7]
```

实验三　　按键输入中断实验

一、实验目的

进一步熟悉 SOPC Builder 和 Nios-II IDE 的使用方法和步骤，掌握 Nios-II 软核的配置方法，掌握对 IO 口的写操作，了解对 Nios II 输入中断的操作；

二、实验模块

FPGA 主控制板模块；

三、实验原理

按键中断时会向 CPU 发出中断申请，CPU 在接到申请后会相应中断，并执行相应的中断服务子程序。

四、实验内容

编写程序，利用按键实现输入中断，每按 button1 次数码管加 1,button2 数码管减 1，实现 0-99 的循环显示；实验者也可以适当修改程序来实现其他效果。

五、实验步骤

1、利用 Quartus II 软件建立工程命名为 Button_intterrupt,启动 SOPC Builder 建立 CPU 命名为 intterrupt；

2、为 CPU 添加 Nios II 软核，选择 Nios II/S,建立片内 4K 的 on-chip ram 命名为 RAM，并为 CPU 添加复位地址为 RAM 地址。添加 PIO 模块，设置为两个 8 路输出，并命名为 LED1、LED2,两个输入 PIO，命名为 BUTTON1，BUTTON2，并设置 BUTTON1 为沿边捕获下降沿中断;然后生成 CPU；

3、建立 File－New－Block Diagram/Schematic File,加入生成的 counter CPU 与输入输出 I/O 口并在重命名后分配管脚，用排线按顺序连接硬件电路。编译后如图 1 所示：

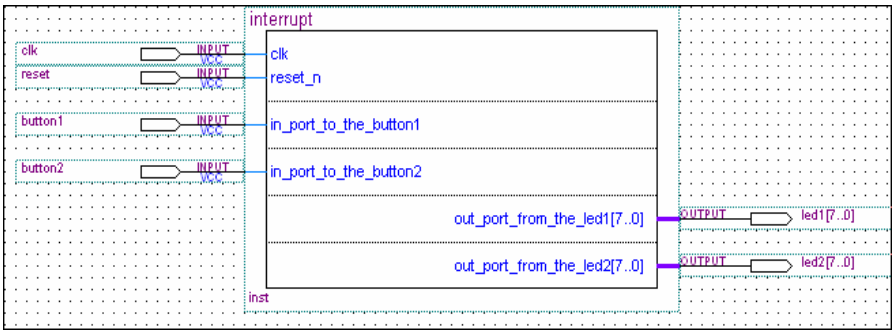


图 4.3.1 顶层原理图

4、启动 Nios II IDE 并重新设置工作目录路径，建立新的空 C/C++应用程序；选择 intterupt.ptf CPU 和命名工程为 intterupt（自定义，无汉字或空格）实例如图 3.2 所示，然后新建 button_interrupt.c

文件，录入参考程序，编译文件。

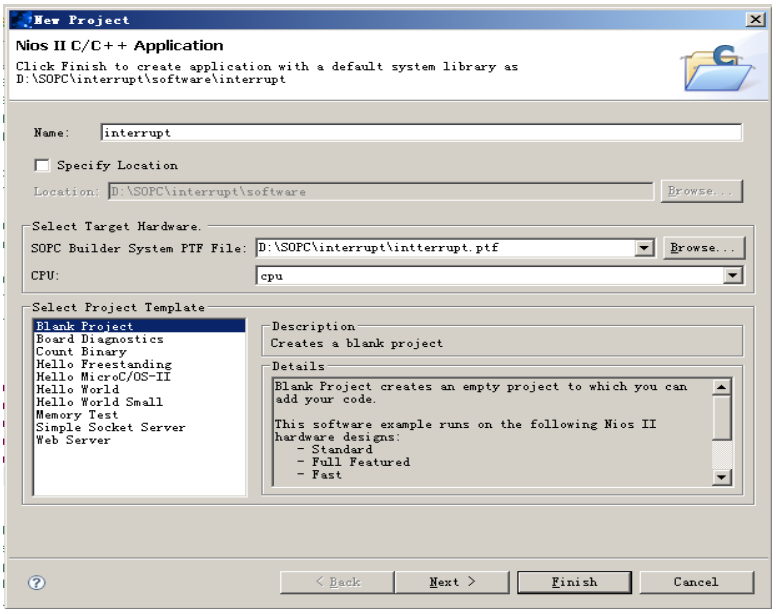


图 4.3.2 新建 interrupt 工程

5、下载。首先启动 Quartus II 下的 Programer,把 Button_intterrupt.sof 文件配置到 FPGA 中，然后在 Nios II IDE 中 Run as→Nios II Hardware,开始运行程序，成功后.按下 BUTTON1 和 BUTTON2 时可以观察到两个数码管的 0-99 的加一减一循环显示。

六、参考程序

```
#include "system.h"
#include "string.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
alt_u8 counter=0;
void delay(alt_u16 temp)
{ alt_u16 i=0;
    while (i<temp)i++;
}
volatile alt_irq_context    g_context;
static void BUTTON1_IRQ_ISR(void * context,alt_u32 id)
    {volatile int* g_context_ptr = (volatile int*) g_context;
   if (counter<99)counter=counter+1;
```

```
        else    counter=0;
        while(!IORD_ALTERA_AVALON_PIO_DATA(BUTTON1_BASE));
        delay(30000);
        *g_context_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON1_BASE);
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON1_BASE, 0);
IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON1_BASE,0x01);
}
static void Init_BUTTON1Ext_interrupt()
{//-- Reset the edge capture register.
void* g_context_ptr = (void*)&g_context;
alt_irq_init(ALT_IRQ_BASE);
IOWR_ALTERA_AVALON_PIO_DIRECTION(BUTTON1_BASE,0x00);//input
IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON1_BASE,0xff);
IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON1_IRQ, 0x0);
alt_irq_register(BUTTON1_IRQ,g_context_ptr,BUTTON1_IRQ_ISR);
}
static void button_not_interruput()
{
 if(IORD_ALTERA_AVALON_PIO_DATA(BUTTON2_BASE)==0x00)
           {     if(counter>0)counter=counter-1;
                else counter=99;
        IOWR_ALTERA_AVALON_PIO_DATA(LED1_BASE, led_table[counter/10]);
        IOWR_ALTERA_AVALON_PIO_DATA(LED2_BASE, led_table[counter%10]);
        while(!IORD_ALTERA_AVALON_PIO_DATA(BUTTON2_BASE));
         delay(30000);
           }
}
int main (void) __attribute__ ((weak, alias ("alt_main")));
int alt_main (void)
{ alt_u8    led_table[10]={0xC0,0XF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
```

```
    Init_BUTTON1Ext_interrupt();
    while (1)
  { button_not_interruput();
OWR_ALTERA_AVALON_PIO_DATA(LED1_BASE, led_table[counter/10]);
    IOWR_ALTERA_AVALON_PIO_DATA(LED2_BASE, led_table[counter%10]);
    delay(20000);
  }
   return 0;
}
```

管脚分配：

```
set_location_assignment PIN_31 -to CLK
set_location_assignment PIN_209 -to RESET
set_location_assignment PIN_182 -to LED1[0]
set_location_assignment PIN_184 -to LED1[1]
set_location_assignment PIN_186 -to LED1[2]
set_location_assignment PIN_188 -to LED1[3]
set_location_assignment PIN_194 -to LED1[4]
set_location_assignment PIN_196 -to LED1[5]
set_location_assignment PIN_198 -to LED1[6]
set_location_assignment PIN_200 -to LED1[7]
set_location_assignment PIN_202 -to LED2[0]
set_location_assignment PIN_207 -to LED2[1]
set_location_assignment PIN_216 -to LED2[2]
set_location_assignment PIN_218 -to LED2[3]
set_location_assignment PIN_220 -to LED2[4]
set_location_assignment PIN_222 -to LED2[5]
set_location_assignment PIN_224 -to LED2[6]
set_location_assignment PIN_230 -to LED2[7]
set_location_assignment PIN_181 -to BUTTON1
set_location_assignment PIN_183 -to BUTTON2
```

实验四　　定时计数器实验

一、实验目的

掌握 SOPC Builder 和 Nios-II IDE 的使用方法和步骤，掌握 Nios-II 软核的配置方法，掌握对 IO 口的读写操作，掌握 Nios II 定时计数器的使用及中断的操作。

二、实验模块

FPGA 主控制板模块；

三、实验原理

定时器工作时是将这2个寄存器的值调入32位计数器，然后根据CPU的时钟，逐步递减计数器的值，直到减到0为止，然后触发中断，并且再次从预制寄存器中将预制值调入32位计数器中设置的初始值，实验者也可以使用下面的2个函数来改变计数器的初始值

IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_0_BASE, TimerValueLow);

IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_0_BASE, TimerValueHigh);

其中TimerValueLow和TimerValueHigh是你要设置的低16位和高16位的定时器初值。计数初值计算方法：计数初值=2^32-FREQ*时间 。

四、实验内容

编写程序，利用定时计数器实现每秒数码管加 1 在 0-99 的循环显示，并半随一个小灯闪烁；实验者也可以适当修改程序来实现其他效果。

五、实验步骤

1、利用 Quartus II 软件建立工程为 timer_intterrupt，启动 SOPC Builder 建立 CPU 命名为 timer；

2、为 CPU 添加 Nios II 软核，选择 Nios II/S,建立片内 8K 的 on-chip ram 命名为 RAM，并为 CPU 添加复位地址为 RAM 地址。添加 PIO 模块，设置为两个 8 路输出，并命名为 LED1、LED2,再设置一个输出 PIO 端口命名为 LED;然后生成 CPU；

3、建立 File－New－Block Diagram/Schematic File,加入生成的 timer CPU 与输入输出 I/O 口，并在重命名后分配管脚，用排线按顺序连接硬件电路。编译后如图 4.1 所示：

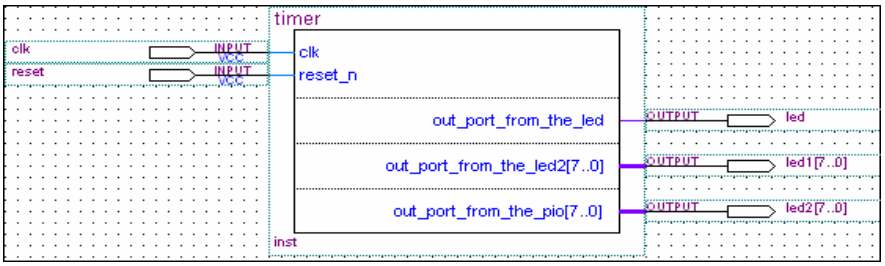


图 4.4.1 顶层原理图

4、启动 Nios II IDE 并重新设置工作目录路径，建立新的空 C/C++应用程序；选择 timer.ptf CPU 和命名工程为 timer（自定义，无汉字或空格）实例如图 4.2 所示，然后新建 timer.c 文件，录入参考程序，编译文件。

5、下载。首先启动 Quartus II 下的 Programer,把 timer_intterrupt.sof 文件配置到 FPGA 中，把两个数码管按照分配的引脚连接好，然后在 Nios II IDE 中 Run as→Nios II Hardware.可以观察到两个数码管的 0-99 的循环显示并伴随 LED 小灯的闪烁。

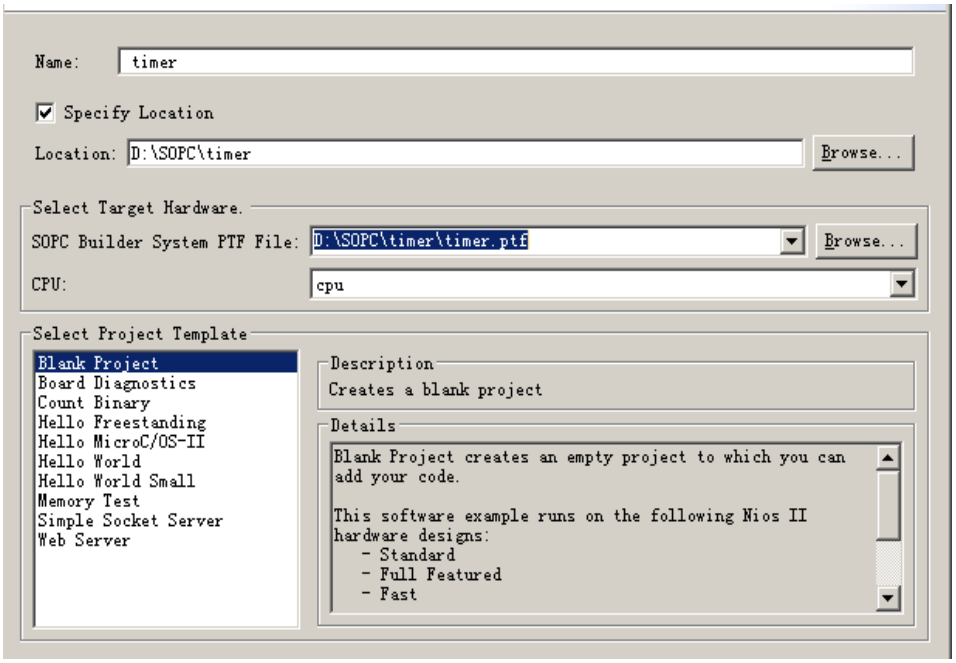


图 4.4.2 新建 Nios II 工程

六、参考程序

```
#include "system.h"
#include "string.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
alt_u8 counter=0;
alt_u8 led_sign=0;
static void handle_Timer1_interrupts(void* context, alt_u32 id)
{IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER1_BASE, 0);//清TO标志
 /****************在下面开始编写用户中断程序****************************/
```

```
 led_sign=~led_sign;
 counter=counter+1;
 IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, led_sign);
 /*****************************************************************/
 }
static void inti_Timer1(void)
{    alt_u8 count=1;
     alt_irq_init(ALT_IRQ_BASE);
     alt_irq_register( TIMER1_IRQ, (void *)& count, handle_Timer1_interrupts); //注册中断函数
IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER1_BASE, 7);//启动timer允许中断连续计数
}
int main (void) __attribute__ ((weak, alias ("alt_main")));
int alt_main (void)
{ alt_u8 led_table[10]={0xC0,0XF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
   inti_Timer1();
   while (1) {IOWR_ALTERA_AVALON_PIO_DATA(LED1_BASE,led_table[(counter%100)/10]);
 IOWR_ALTERA_AVALON_PIO_DATA(LED2_BASE, led_table[counter%10]);
}
   return 0;
}
```

管脚分配：

```
set_location_assignment PIN_31 -to CLK
set_location_assignment PIN_209 -to RESET
set_location_assignment PIN_182 -to LED1[0]
set_location_assignment PIN_184 -to LED1[1]
set_location_assignment PIN_186 -to LED1[2]
set_location_assignment PIN_188 -to LED1[3]
set_location_assignment PIN_194 -to LED1[4]
set_location_assignment PIN_196 -to LED1[5]
```

```
set_location_assignment PIN_198 -to LED1[6]
set_location_assignment PIN_200 -to LED1[7]
set_location_assignment PIN_202 -to LED2[0]
set_location_assignment PIN_207 -to LED2[1]
set_location_assignment PIN_216 -to LED2[2]
set_location_assignment PIN_218 -to LED2[3]
set_location_assignment PIN_220 -to LED2[4]
set_location_assignment PIN_222 -to LED2[5]
set_location_assignment PIN_224 -to LED2[6]
set_location_assignment PIN_230 -to LED2[7]
set_location_assignment PIN_181 -to LED
```

实验五　串行口通信实验

一、实验目的

熟练掌握 SOPC Builder 和 Nios-II IDE 的使用方法和步骤，Nios-II 软核的配置方法，初步学会 Nios II 串口的使用及其中断的操作方法。

二、实验模块

FPGA 主控制板模块；串口通信模块。

三、实验原理

利用串口控制核，向串口里发送数据，并通过串口调试软件观察接收到的数据。

四、实验内容

编写程序，借助于串口调试工具或 JTAG UART，实现串口中断接收 PC 数据后再由串口发送回 PC 机。

五、实验步骤

1、利用 Quartus II 软件建立工程命名为 uart_intterrupt,启动 SOPC Builder 建立 CPU 命名为 uart。

2、为 CPU 添加 Nios II 软核，选择 Nios II/S,建立片内 4K 的 on-chip ram 命名为 RAM，并为 CPU 添加复位地址为 RAM 地址。添加 UART 模块设置波特率为 115200；8 位数据位，无校验位，1 停止位;然后生成 CPU。

3、建立 File－New－Block Diagram/Schematic File,加入生成的 uart CPU 与输入输出 I/O 口，并在重命名后分配管脚，用排线按顺序连接硬件电路，编译后如图 5.1 所示。

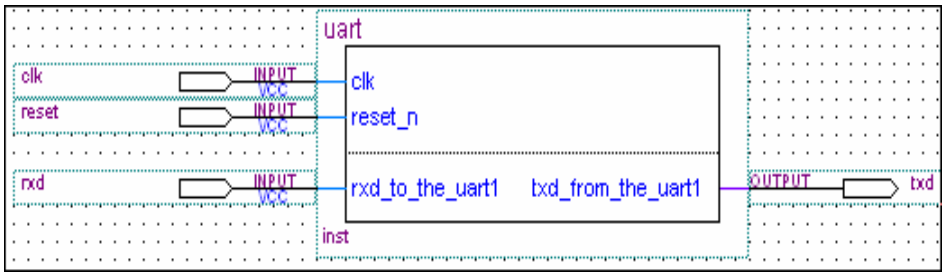


图 4.5.1 顶层原理图

4、启动 Nios II IDE 并重新设置工作目录路径，建立新的空 C/C++应用程序；选择 UART.ptf CPU 和命名工程为 uart（自定义，无汉字或空格）实例如图 4.2 所示，然后新建 uart.c 文件，录入参考程序，编译文件。

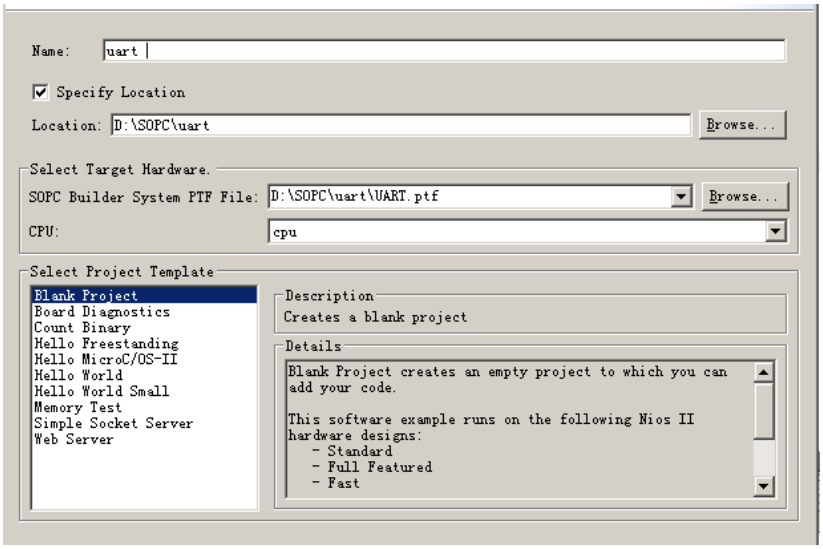


图 4.5.2 新建 Nios II 工程

5、下载。首先启动 Quartus II 下的 Programer,把 uart_intterrupt.sof 文件配置到 FPGA 中，再连接 FPGA 和 RS232 串口模块，使用串口线接到电脑的串口端，使用串口调试工具发送数据来判断程序的正确性。

然后在 Nios II IDE 中右键选择 Run as→Nios II Hardware.由串口调试助手或 JTAG UART0 来观察发送和接收数据。

六、参考程序

```
#include "system.h"
#include "string.h"
#include "altera_avalon_uart_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
void uart_WRData(unsigned char data )
 { //等待发送寄存器为空
 while(!(IORD_ALTERA_AVALON_UART_STATUS(UART0_BASE)&0X20));
 IOWR_ALTERA_AVALON_UART_TXDATA(UART0_BASE, data);
 }
static void handle_UART0_interrupts(void* context, alt_u32 id)
{    alt_u16 data;
 /****************在下面开始编写用户中断处理程*************************/
      data =IORD_ALTERA_AVALON_UART_RXDATA(UART0_BASE);
```

```
        uart_WRData(data);
/*****************************************************************/
 }
static void inti_UART0(void)
{ alt_u8 count=1;
   alt_irq_init(ALT_IRQ_BASE);
   alt_irq_register( UART0_IRQ, (void *)& count, handle_UART0_interrupts); //注册中断函数
   IOWR_ALTERA_AVALON_UART_CONTROL(UART0_BASE,0x080); //允许接收完成中断
}
int main (void) __attribute__((weak, alias ("alt_main")));
int alt_main (void)
{ inti_UART0();
   while(1);
 return 0;
}
```

七、试验分析

修改程序，实现字符串的发送和接收。并使用串口调试工具验证。

管脚分配：

```
set_location_assignment PIN_31 -to CLK
set_location_assignment PIN_209 -to RESET
set_location_assignment PIN_181 -to RXD
set_location_assignment PIN_183 -to TXD
```

实验六　　存储器配置实验

一、实验目的

1、 进一步熟悉 SOPC 系统的构成，掌握完整的 NIOS II CPU 的定制；

2、 熟悉锁相环的使用及配置方法。

二、实验任务

SOPC 系统定制 FLASH(TE28F320J3A110_BYTE)、SDRAM(single Micron MT48LC4M32B2-7 chip),编程实现 Hello word！及流水灯效果。

三、实验步骤

1、 建立工程命名为 Project，启动 SOPC Builder。

2、 添加 Nios II 处理器，在存储器或存储器控制器（memeries and memory contorller）中添加 SDRAM contoller，如图 1 所示：

图 1. 选择 SDRAM 类型

3、 同上继续添加 FLASH controller，型号选择 TE28F320J3A110_BYTE

4、 添加三态桥总线用于 FLASH 的控制，选择如图 2 所示：

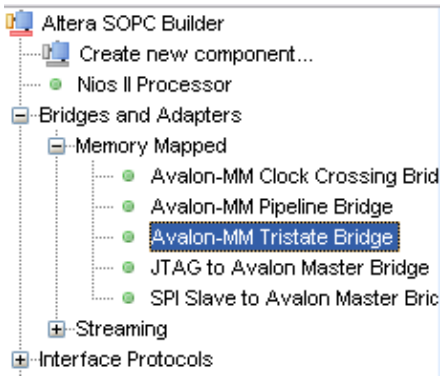


图 2 选择 Avalon-MM 三态桥

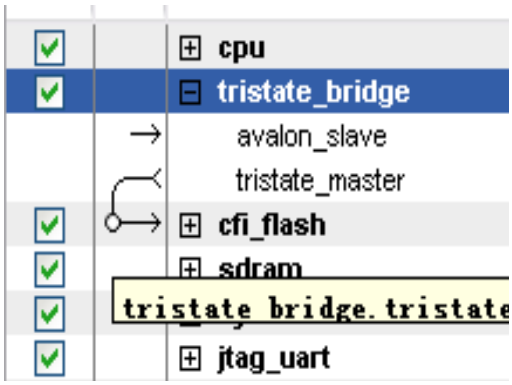


图 3 连接 FLASH 和三态桥控制总线

5、 添加 jtag_uart，和 8 位 PIO，用来实现实验效果，生成 CPU。

6、 新建原理图文件命名为 Project.bdf，新建 PLL 锁相环，输入时钟 25M，输出时钟 CLK050M，CLK1 设置 50M 偏移量-126deg。如图 4，5 所示。

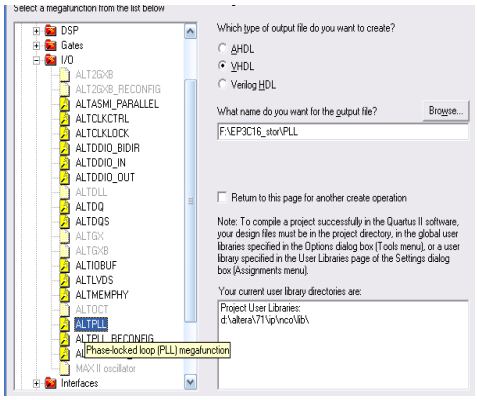


图 4、新建 PLL 锁相环

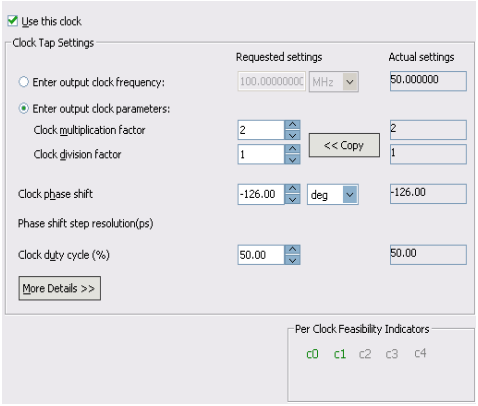
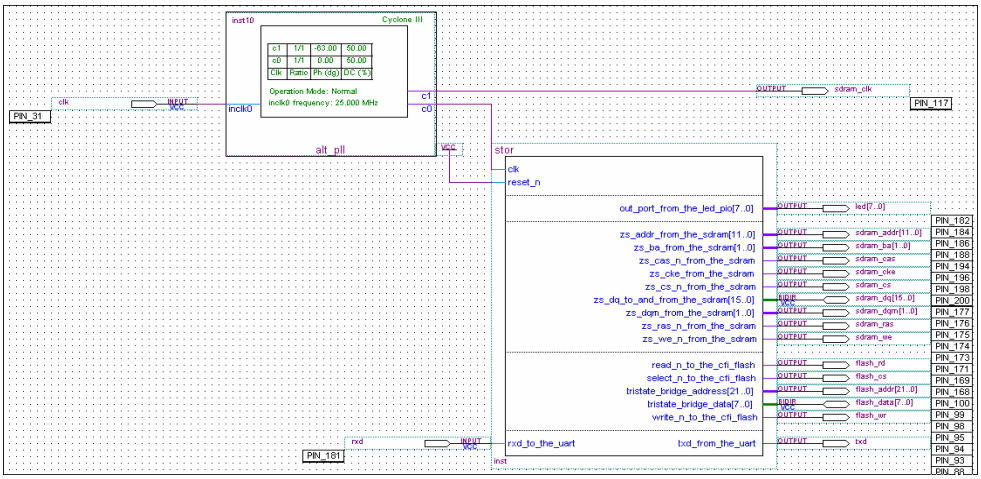


图 5、设置 SDRAM 时钟偏移量

7、连接并分配管脚，保存后，全编译。



8、启动 Nios II IDE，新建工程如图 6 所示

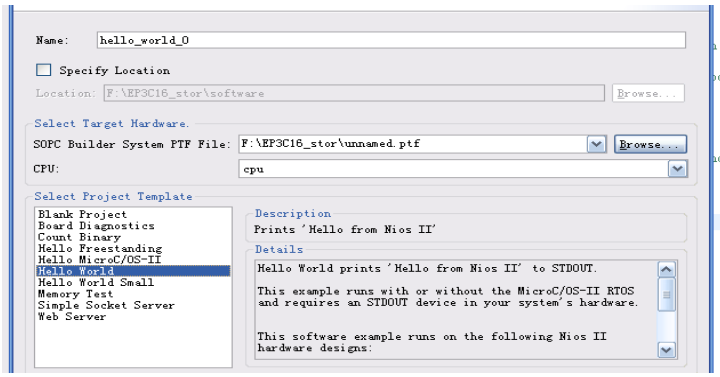


图 6 新建 Nios II ide 工程

四、实验参考程序：

```
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
```

```
    void delay(unsigned int del)
    {
     while(del)del--;
      }
  int main()
  {unsigned int i,dat;
  printf("Hello from Nios II!\n");
 while(1)
 {dat=0x7f;
  for(i=0;i<8;i++)
  {
   IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE,dat);
   dat=dat>>1;
   delay(200000);
 }
  printf("Finish one time cycle!\n");
}
 return 0;
}
```

管脚配置：

set_location_assignment PIN_31 -to clk

set_location_assignment PIN_181 -to rxd

set_location_assignment PIN_183 -to txd

set_location_assignment PIN_85 -to flash_addr[21]

set_location_assignment PIN_84 -to flash_addr[20]

set_location_assignment PIN_83 -to flash_addr[19]

set_location_assignment PIN_82 -to flash_addr[18]

set_location_assignment PIN_81 -to flash_addr[17]

set_location_assignment PIN_80 -to flash_addr[16]

set_location_assignment PIN_78 -to flash_addr[15]

set_location_assignment PIN_51 -to flash_addr[14]

set_location_assignment PIN_52 -to flash_addr[13]

set_location_assignment PIN_55 -to flash_addr[12]

set_location_assignment PIN_57 -to flash_addr[11]

set_location_assignment PIN_63 -to flash_addr[10]

set_location_assignment PIN_64 -to flash_addr[9]

set_location_assignment PIN_65 -to flash_addr[8]

set_location_assignment PIN_68 -to flash_addr[7]

set_location_assignment PIN_69 -to flash_addr[6]

set_location_assignment PIN_70 -to flash_addr[5]

set_location_assignment PIN_71 -to flash_addr[4]

set_location_assignment PIN_72 -to flash_addr[3]

set_location_assignment PIN_73 -to flash_addr[2]

set_location_assignment PIN_76 -to flash_addr[1]

set_location_assignment PIN_49 -to flash_addr[0]

set_location_assignment PIN_56 -to flash_cs

set_location_assignment PIN_37 -to flash_data[7]

set_location_assignment PIN_38 -to flash_data[6]

set_location_assignment PIN_39 -to flash_data[5]

set_location_assignment PIN_41 -to flash_data[4]

set_location_assignment PIN_43 -to flash_data[3]

set_location_assignment PIN_44 -to flash_data[2]

set_location_assignment PIN_45 -to flash_data[1]

set_location_assignment PIN_46 -to flash_data[0]

set_location_assignment PIN_22 -to flash_rd

set_location_assignment PIN_21 -to flash_wr

set_location_assignment PIN_160 -to sdram_addr[11]

set_location_assignment PIN_147 -to sdram_addr[10]

set_location_assignment PIN_103 -to sdram_addr[9]

set_location_assignment PIN_106 -to sdram_addr[8]

set_location_assignment PIN_107 -to sdram_addr[7]

set_location_assignment PIN_108 -to sdram_addr[6]

set_location_assignment PIN_109 -to sdram_addr[5]

set_location_assignment PIN_110 -to sdram_addr[4]

set_location_assignment PIN_111 -to sdram_addr[3]

set_location_assignment PIN_144 -to sdram_addr[2]

set_location_assignment PIN_145 -to sdram_addr[1]

set_location_assignment PIN_146 -to sdram_addr[0]

set_location_assignment PIN_148 -to sdram_ba[1]

set_location_assignment PIN_159 -to sdram_ba[0]

set_location_assignment PIN_164 -to sdram_cas

set_location_assignment PIN_102 -to sdram_cke

set_location_assignment PIN_161 -to sdram_cs

set_location_assignment PIN_101 -to sdram_dqm[1]

set_location_assignment PIN_167 -to sdram_dqm[0]

set_location_assignment PIN_162 -to sdram_ras

set_location_assignment PIN_166 -to sdram_we

set_location_assignment PIN_117 -to sdram_clk

set_location_assignment PIN_177 -to sdram_dq[0]

set_location_assignment PIN_176 -to sdram_dq[1]

set_location_assignment PIN_175 -to sdram_dq[2]

set_location_assignment PIN_174 -to sdram_dq[3]

set_location_assignment PIN_173 -to sdram_dq[4]

set_location_assignment PIN_171 -to sdram_dq[5]

set_location_assignment PIN_169 -to sdram_dq[6]

set_location_assignment PIN_168 -to sdram_dq[7]

set_location_assignment PIN_100 -to sdram_dq[8]

set_location_assignment PIN_99 -to sdram_dq[9]

set_location_assignment PIN_98 -to sdram_dq[10]

set_location_assignment PIN_95 -to sdram_dq[11]

set_location_assignment PIN_94 -to sdram_dq[12]

set_location_assignment PIN_93 -to sdram_dq[13]

set_location_assignment PIN_88 -to sdram_dq[14]

set_location_assignment PIN_87 -to sdram_dq[15]

set_location_assignment PIN_142 -to sdram_dq[16]

set_location_assignment PIN_139 -to sdram_dq[17]

set_location_assignment PIN_137 -to sdram_dq[18]

set_location_assignment PIN_135 -to sdram_dq[19]

set_location_assignment PIN_134 -to sdram_dq[20]

set_location_assignment PIN_133 -to sdram_dq[21]

set_location_assignment PIN_132 -to sdram_dq[22]

set_location_assignment PIN_131 -to sdram_dq[23]

set_location_assignment PIN_128 -to sdram_dq[24]

set_location_assignment PIN_127 -to sdram_dq[25]

set_location_assignment PIN_126 -to sdram_dq[26]

set_location_assignment PIN_120 -to sdram_dq[27]

set_location_assignment PIN_119 -to sdram_dq[28]

set_location_assignment PIN_118 -to sdram_dq[29]

set_location_assignment PIN_114 -to sdram_dq[30]

set_location_assignment PIN_113 -to sdram_dq[31]

set_location_assignment PIN_143 -to sdram_dqm[2]

set_location_assignment PIN_112 -to sdram_dqm[3]

set_location_assignment PIN_182 -to led[0]

set_location_assignment PIN_184 -to led[1]

set_location_assignment PIN_186 -to led[2]

set_location_assignment PIN_188 -to led[3]

set_location_assignment PIN_194 -to led[4]

set_location_assignment PIN_196 -to led[5]

set_location_assignment PIN_198 -to led[6]

set_location_assignment PIN_200 -to led[7]

实验七　　4 乘 4 键盘实验

一、实验目的

1、了解 4X4 键盘的工作原理

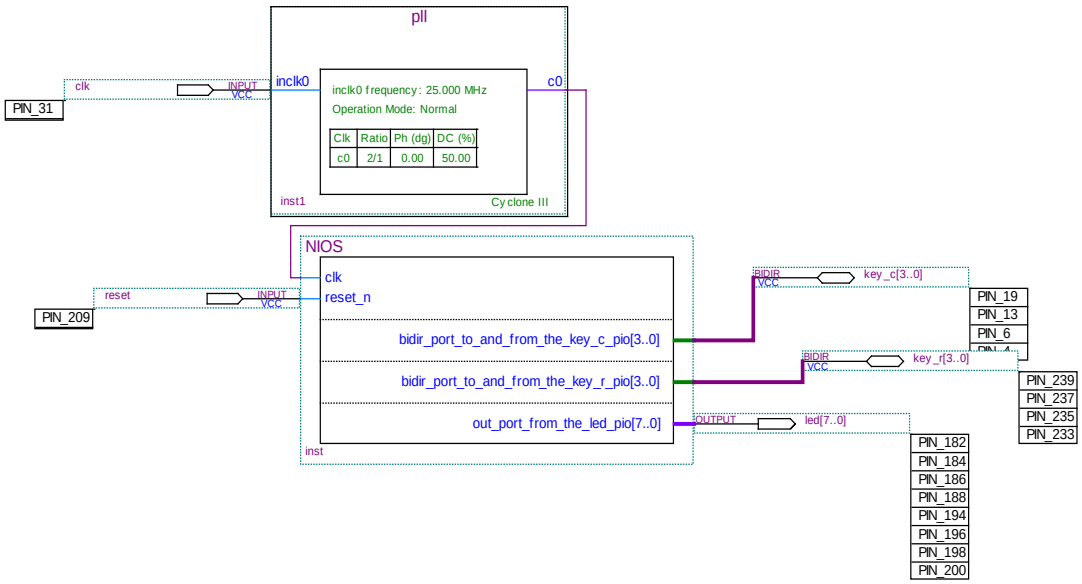
2、进一步学习在 Quartus 和 NIos 环境下进行试验。

二、实验原理

4 乘 4 键盘可以输出从 1—F 的数字显示，通过按键，在数码管上显示相应的数值。键盘的扫描一般有两种方法：反转法和逐行扫描法。本实验采用反转法，先将行置 0，扫描列；然后再将列置 0，扫描行。

三、实验步骤

1、新建工程文件，命名为 jianpan，打开 SOPC 工具，添加 nios 软核，生成 block 文件，编译后下载。如下图所示。



2、打开 Nios II IDE, 新建工程，选择相应的 CPU 后编写代码，编译后下载。

四、参考程序

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#include <stdio.h>

unsigned char num[4][4]={{0xc0,0xf9,0xa4,0xb0},
                         {0x99,0x92,0x82,0xf8},
                         {0x80,0x90,0x88,0x83},
                         {0xc6,0xa1,0x86,0x8e}};
```

```
void delay(unsigned char del)
{while(del--);}
void main(void)
{
 unsigned char i=0,j=0;
 unsigned char hang,lie;
 while(1)
 {
  IOWR_ALTERA_AVALON_PIO_DIRECTION(KEY_R_PIO_BASE,0xF);
  IOWR_ALTERA_AVALON_PIO_DIRECTION(KEY_C_PIO_BASE,0x0);
  IOWR_ALTERA_AVALON_PIO_DATA(KEY_R_PIO_BASE,0x0);
  delay(500);
  lie=IORD_ALTERA_AVALON_PIO_DATA(KEY_C_PIO_BASE);
  if(lie!=0xF)
  {
   switch(lie)
   {
    case 0xe:   i=0;  break;
    case 0xd:   i=1;  break;
    case 0xb:   i=2;  break;
    case 0x7:   i=3;  break;
    default:    i=4;  break;
   }
  IOWR_ALTERA_AVALON_PIO_DIRECTION(KEY_R_PIO_BASE,0x0);
  IOWR_ALTERA_AVALON_PIO_DIRECTION(KEY_C_PIO_BASE,0xF);
  IOWR_ALTERA_AVALON_PIO_DATA(KEY_C_PIO_BASE,0x0);
  delay(500);
  hang=IORD_ALTERA_AVALON_PIO_DATA(KEY_R_PIO_BASE);
  if(hang!=0xF)
  {
   switch(hang)
   {
    case 0x7:   j=3;  break;
    case 0xb:   j=2;  break;
    case 0xd:   j=1;  break;
    case 0xe:   j=0;  break;
    default:    i=4;  break;
   }
  }
 }
 printf("%x %x\n",hang,lie);
 IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE,num[j][i]);
}
}
```

管脚分配：

```
set_location_assignment PIN_19 -to key_c[3]
set_location_assignment PIN_13 -to key_c[2]
set_location_assignment PIN_6 -to key_c[1]
set_location_assignment PIN_4 -to key_c[0]
set_location_assignment PIN_239 -to key_r[3]
set_location_assignment PIN_237 -to key_r[2]
set_location_assignment PIN_235 -to key_r[1]
set_location_assignment PIN_233 -to key_r[0]
set_location_assignment PIN_209 -to reset
set_location_assignment PIN_182 -to led[0]
set_location_assignment PIN_184 -to led[1]
set_location_assignment PIN_186 -to led[2]
set_location_assignment PIN_188 -to led[3]
set_location_assignment PIN_194 -to led[4]
set_location_assignment PIN_196 -to led[5]
set_location_assignment PIN_198 -to led[6]
set_location_assignment PIN_200 -to led[7]
set_location_assignment PIN_31 -to clk
```

第五章 SOPC系统综合实验

实验一 高速DAC实验

一、实验目的

进一步熟练使用 SOPC 软件及其外设配置方法，理解 DA 转换原理，掌握模数转电压的计算方法。

二、实验模块

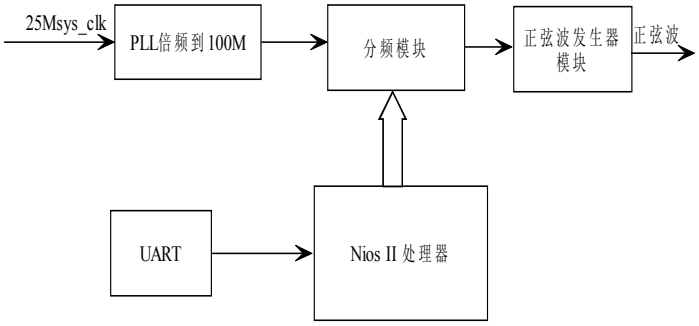
FPGA 主控制板模块；DAC 数模转换模块；RS232 串口通信模块。

三、实验原理

本实验采用美国ADI公司出品的视频模数转换电路芯片ADV9767。ADV9767 在单芯片上集成了 2 个独立的 14 位高速D/A转换器，特别适用于高分辨率模拟接口的显示终端和要求高速D／A转换的应用系统。在每个时钟周期的上升沿，三路数据寄存器锁存各自的 14 位视频数据，通过各自的DAC单元电路转换成相应的模拟电流信号，后通过对地电阻转换为电压信号。

四、实验内容

参考 vhdl 基本实验部分的 D/A 接口实验（实验十五），编写程序，使用 PC 机软件通过键盘来设置 DAC 模块的正弦波频率输出。使用示波器验证频率数据的正确性，误差分析后通过软件算法来矫正。实验框图如图



实验框图

如图所示 100M 的时钟信号，经过分频模块，为正弦波发生器提供时钟，Nios II 通过 uart 控制分频系数（0-255，本实验要求实现频率的加减调节。

五、实验步骤

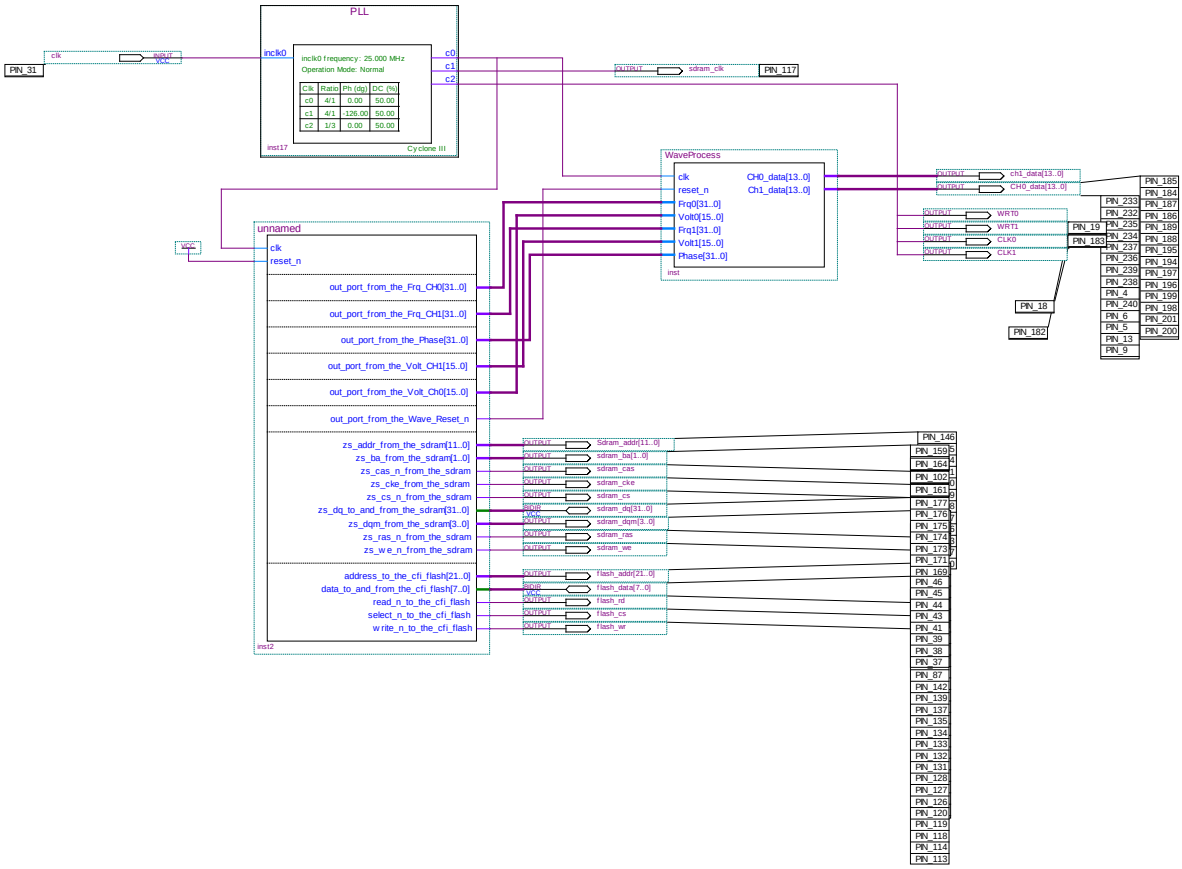
1、利用 Quartus II 软件建立工程命名为 DAC，按照 EDA 实验部分的（实验十五）要求建立正弦波数据表及其发生的原理图，完成正弦波发生器的设计。

2、加入 compare, sub, NCO, PLL 等宏模块。

3、启动 SOPC Builder 建立 CPU。

4、为 CPU 添加 Nios II 软核，选择 Nios II/S,建立片内 8K 的 on-chip ram 命名为 RAM，并为 CPU 添加复位地址为 RAM 地址，加入 sdram 和 flash，然后生成 CPU。

5、加入生成的 CPU 与输入输出 I/O 口，正弦信号发生模块，并在重命名后分配管脚，如图所示，用排线按顺序连接硬件电路，编译。



顶层视图

6、启动 Nios II IDE 并重新设置工作目录路径，建立新的 C/C++应用程序；选择 DA CPU 和空白实例，并录入代码，然后编译文件。

7、下载。首先启动 Quartus II 下的 Programer，把 DAC.sof 文件配置到 FPGA 中，然后在 Nios II IDE 中 Run as→Nios II Hardware.，通过串口调试工具由 UART 设定后使用示波器或逻辑分析仪来观察波形变化。

8、根据例程 SOPC→AdvancedLab→DAC 例程自己动手完成对频率变化特性的分析。

六、参考程序

```
#include "system.h"
#include <io.h>
```

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#define Set_Volt_CH0(dat)     IOWR_ALTERA_AVALON_PIO_DATA(VOLT_CH0_BASE,(dat))
                                                                    //0-65536
#define Set_Volt_CH1(dat)     IOWR_ALTERA_AVALON_PIO_DATA(VOLT_CH1_BASE,(dat))
#define Set_Phase(dat)    IOWR_ALTERA_AVALON_PIO_DATA(PHASE_BASE,(dat))
void SetFrq_ch0(unsigned long dat)
{
    alt_u32    SinFrq;
    SinFrq=0xffffffff*dat/100000000;
    IOWR_ALTERA_AVALON_PIO_DATA(FRQ_CH0_BASE,SinFrq);
}
void SetFrq_ch1(unsigned long dat)
{
    alt_u32    SinFrq;
    SinFrq=0xffffffff*dat/100000000;
    IOWR_ALTERA_AVALON_PIO_DATA(FRQ_CH1_BASE,SinFrq);
}
void Run(void)
{
    IOWR_ALTERA_AVALON_PIO_DATA(WAVE_RESET_N_BASE,1);
}
void Stop(void)
{
    IOWR_ALTERA_AVALON_PIO_DATA(WAVE_RESET_N_BASE,0);
}
#include <stdio.h>
int main()
{
```

```
    printf("Hello from Nios II!\n");
    Run();
    SetFrq_ch0(1000000000);
    Set_Volt_CH0(40000);
    Set_Phase(100);
    return 0;
}
```

管脚分配：

```
    set_location_assignment PIN_31 -to clk
    set_location_assignment PIN_49 -to flash_addr[0]
    set_location_assignment PIN_76 -to flash_addr[1]
    set_location_assignment PIN_73 -to flash_addr[2]
    set_location_assignment PIN_72 -to flash_addr[3]
    set_location_assignment PIN_71 -to flash_addr[4]
    set_location_assignment PIN_70 -to flash_addr[5]
    set_location_assignment PIN_69 -to flash_addr[6]
    set_location_assignment PIN_68 -to flash_addr[7]
    set_location_assignment PIN_65 -to flash_addr[8]
    set_location_assignment PIN_64 -to flash_addr[9]
    set_location_assignment PIN_63 -to flash_addr[10]
    set_location_assignment PIN_57 -to flash_addr[11]
    set_location_assignment PIN_55 -to flash_addr[12]
    set_location_assignment PIN_52 -to flash_addr[13]
    set_location_assignment PIN_51 -to flash_addr[14]
    set_location_assignment PIN_78 -to flash_addr[15]
    set_location_assignment PIN_80 -to flash_addr[16]
    set_location_assignment PIN_81 -to flash_addr[17]
    set_location_assignment PIN_82 -to flash_addr[18]
    set_location_assignment PIN_83 -to flash_addr[19]
    set_location_assignment PIN_84 -to flash_addr[20]
```

```
set_location_assignment PIN_85 -to flash_addr[21]
set_location_assignment PIN_56 -to flash_cs
set_location_assignment PIN_46 -to flash_data[0]
set_location_assignment PIN_45 -to flash_data[1]
set_location_assignment PIN_44 -to flash_data[2]
set_location_assignment PIN_43 -to flash_data[3]
set_location_assignment PIN_41 -to flash_data[4]
set_location_assignment PIN_39 -to flash_data[5]
set_location_assignment PIN_38 -to flash_data[6]
set_location_assignment PIN_37 -to flash_data[7]
set_location_assignment PIN_22 -to flash_rd
set_location_assignment PIN_21 -to flash_wr
set_location_assignment PIN_209 -to reset_n
set_location_assignment PIN_146 -to Sdram_addr[0]
set_location_assignment PIN_145 -to Sdram_addr[1]
set_location_assignment PIN_144 -to Sdram_addr[2]
set_location_assignment PIN_111 -to Sdram_addr[3]
set_location_assignment PIN_110 -to Sdram_addr[4]
set_location_assignment PIN_109 -to Sdram_addr[5]
set_location_assignment PIN_108 -to Sdram_addr[6]
set_location_assignment PIN_107 -to Sdram_addr[7]
set_location_assignment PIN_106 -to Sdram_addr[8]
set_location_assignment PIN_103 -to Sdram_addr[9]
set_location_assignment PIN_147 -to Sdram_addr[10]
set_location_assignment PIN_160 -to Sdram_addr[11]
set_location_assignment PIN_159 -to sdram_ba[0]
set_location_assignment PIN_148 -to sdram_ba[1]
set_location_assignment PIN_164 -to sdram_cas
set_location_assignment PIN_102 -to sdram_cke
set_location_assignment PIN_117 -to sdram_clk
```

```
set_location_assignment PIN_161 -to sdram_cs
set_location_assignment PIN_177 -to sdram_dq[0]
set_location_assignment PIN_176 -to sdram_dq[1]
set_location_assignment PIN_175 -to sdram_dq[2]
set_location_assignment PIN_174 -to sdram_dq[3]
set_location_assignment PIN_173 -to sdram_dq[4]
set_location_assignment PIN_171 -to sdram_dq[5]
set_location_assignment PIN_169 -to sdram_dq[6]
set_location_assignment PIN_168 -to sdram_dq[7]
set_location_assignment PIN_100 -to sdram_dq[8]
set_location_assignment PIN_99 -to sdram_dq[9]
set_location_assignment PIN_98 -to sdram_dq[10]
set_location_assignment PIN_95 -to sdram_dq[11]
set_location_assignment PIN_94 -to sdram_dq[12]
set_location_assignment PIN_93 -to sdram_dq[13]
set_location_assignment PIN_88 -to sdram_dq[14]
set_location_assignment PIN_87 -to sdram_dq[15]
set_location_assignment PIN_142 -to sdram_dq[16]
set_location_assignment PIN_139 -to sdram_dq[17]
set_location_assignment PIN_137 -to sdram_dq[18]
set_location_assignment PIN_135 -to sdram_dq[19]
set_location_assignment PIN_134 -to sdram_dq[20]
set_location_assignment PIN_133 -to sdram_dq[21]
set_location_assignment PIN_132 -to sdram_dq[22]
set_location_assignment PIN_131 -to sdram_dq[23]
set_location_assignment PIN_128 -to sdram_dq[24]
set_location_assignment PIN_127 -to sdram_dq[25]
set_location_assignment PIN_126 -to sdram_dq[26]
set_location_assignment PIN_120 -to sdram_dq[27]
set_location_assignment PIN_119 -to sdram_dq[28]
```

```
set_location_assignment PIN_118 -to sdram_dq[29]
set_location_assignment PIN_114 -to sdram_dq[30]
set_location_assignment PIN_113 -to sdram_dq[31]
set_location_assignment PIN_167 -to sdram_dqm[0]
set_location_assignment PIN_101 -to sdram_dqm[1]
set_location_assignment PIN_143 -to sdram_dqm[2]
set_location_assignment PIN_112 -to sdram_dqm[3]
set_location_assignment PIN_162 -to sdram_ras
set_location_assignment PIN_166 -to sdram_we
set_location_assignment PIN_233 -to CH0_data[0]
set_location_assignment PIN_232 -to CH0_data[1]
set_location_assignment PIN_235 -to CH0_data[2]
set_location_assignment PIN_234 -to CH0_data[3]
set_location_assignment PIN_237 -to CH0_data[4]
set_location_assignment PIN_236 -to CH0_data[5]
set_location_assignment PIN_239 -to CH0_data[6]
set_location_assignment PIN_238 -to CH0_data[7]
set_location_assignment PIN_4 -to CH0_data[8]
set_location_assignment PIN_240 -to CH0_data[9]
set_location_assignment PIN_6 -to CH0_data[10]
set_location_assignment PIN_5 -to CH0_data[11]
set_location_assignment PIN_13 -to CH0_data[12]
set_location_assignment PIN_9 -to CH0_data[13]
set_location_assignment PIN_185 -to ch1_data[0]
set_location_assignment PIN_184 -to ch1_data[1]
set_location_assignment PIN_187 -to ch1_data[2]
set_location_assignment PIN_186 -to ch1_data[3]
set_location_assignment PIN_189 -to ch1_data[4]
set_location_assignment PIN_188 -to ch1_data[5]
set_location_assignment PIN_195 -to ch1_data[6]
```

```
set_location_assignment PIN_194 -to ch1_data[7]
set_location_assignment PIN_197 -to ch1_data[8]
set_location_assignment PIN_196 -to ch1_data[9]
set_location_assignment PIN_199 -to ch1_data[10]
set_location_assignment PIN_198 -to ch1_data[11]
set_location_assignment PIN_201 -to ch1_data[12]
set_location_assignment PIN_200 -to ch1_data[13]
set_location_assignment PIN_18 -to CLK0
set_location_assignment PIN_183 -to CLK1
set_location_assignment PIN_19 -to WRT0
set_location_assignment PIN_182 -to WRT1
```

实验二 DDS实验

一、实验目的

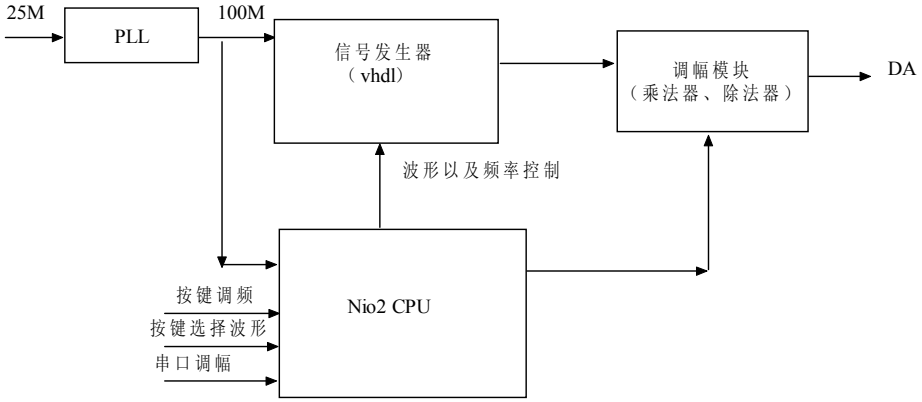
进一步熟练使用 SOPC 软件及其配置方法，理解直接数字频率合成(DDS)实验原理，掌握其实现方法。

二、实验模块

FPGA 主控制板模块；DAC 数模转换模块；RS232 串口通信模块。

三、实验原理

DDS 系统的核心是相位累加器，每来一个时钟脉冲，它的内容就更新一次。在每次更新时，相位增量寄存器的相位增量 M 就加到相位累加器中的相位累加值上。相位累加器的输出作为正弦查找表的查找地址。查找表中的每个地址代表一个周期的正弦波的一个相位点，每个相位点对应一个量化振幅值。因此，这个查找表相当于一个相位／振幅变换器，它将相位累加器的相位信息映射成数字振幅信息，这个数字振幅值就作为 D/A 变换器的输入。



实验框图

四、实验内容

编写程序，实现正弦波、方波、三角波、锯齿波的输出，要求波形频率,幅度在一定范围内连续可调。并且通过按键或串口能够进行相应的调节。输出接高速 DA。参考程序：详见参考例程

五、实验步骤

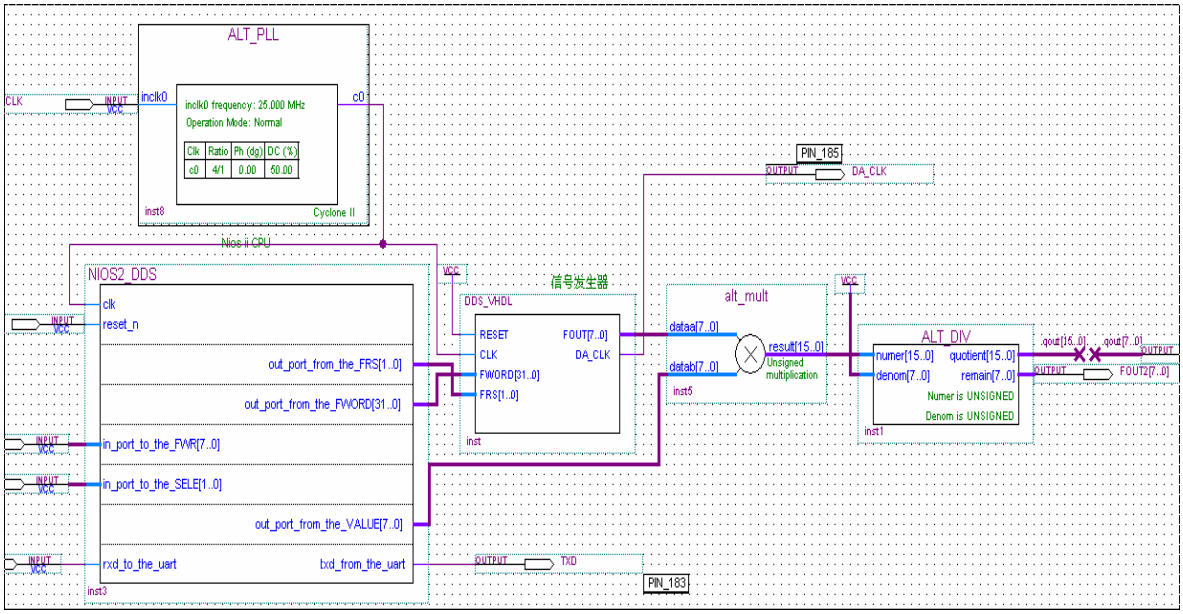
1、利用 Quartus II 软件建立工程命名为 DDS。

2、波形的产生：正弦波：利用 IP 核建立 NCO，并设置为 8 位频率控制字，8 位相位控制字，8 位波形数据输出。方波：利用 8 位计数器根据 8 位的频率控制字来计数并输出相应得 8 位 0 或 8 位 1。三角波：利用一个计数器 2 先由 0 到 255 计数再有 255 到 0 反相计数产生方波。锯齿波：利用计数器 3 由 15 计数到 255 再由 15 开始重新计数产生锯齿波。

3、启动 SOPC Builder 建立 CPU 命名 Nios2_DDS；为 CPU 添加 Nios II 软核，建立片内 4K

的 on-chip ram 命名为 RAM，并为 CPU 添加复位地址为 RAM 地址。添加 UART 模块设置波特率为 115200；8 位数据位，无校验位，1 停止位；添加波形转换和频率控制字输入输出端口 PIO;然后生成 CPU；

4、建立 File－New－Block Diagram/Schematic File,设置锁相环输出 100M 的时钟；加入生成的 DA CPU 与输入输出 I/O 口，添加并设置除法器（LPM_DIV）,乘法器（LPM_MULT），分别命名为 ALT_DIV，ALT_MULT；重命名后分配管脚，用排线连接硬件电路，如图所示。编译。

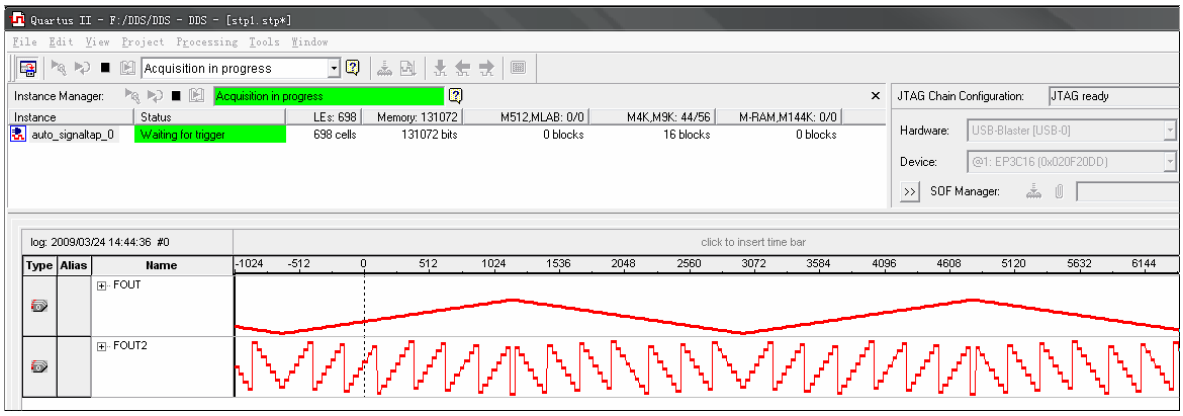


顶层图

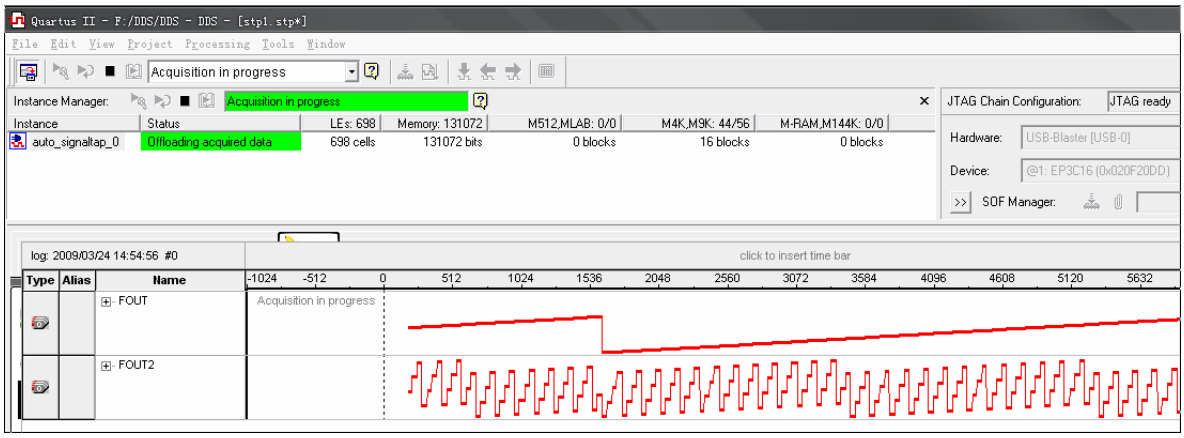
5、新建 Nios2IDE 工程，及工程文件，实现实时读取按键数据，并接收串口调幅数据。编译。

6、下载。首先启动 Quartus II 下的 Programer，把 DDS.sof 文件配置到 FPGA 中，然后在 Nios II IDE 中 Run as---Nios II Hardware。

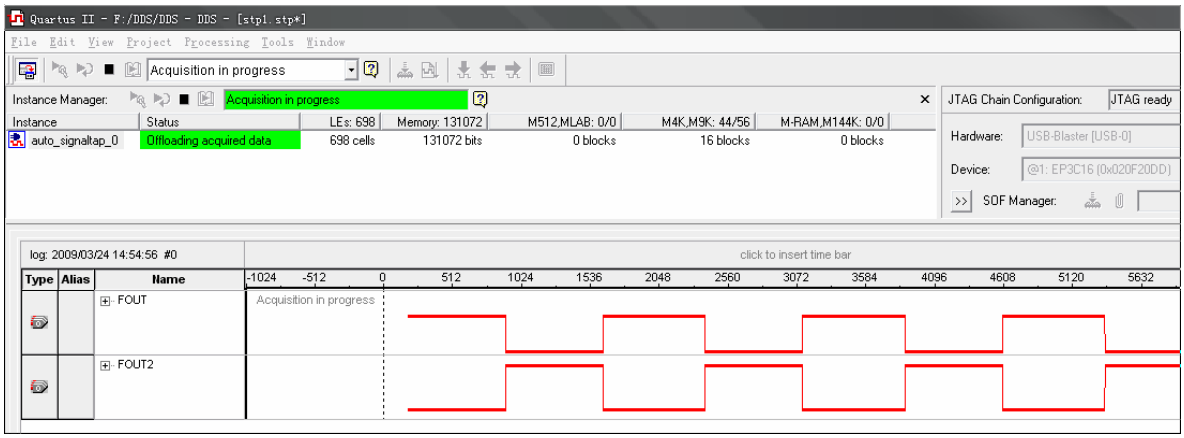
7、运行成功后，.由按键来控制波形的选择，以及频率的大小，PC 机通过串口调试工具来设置波形的幅度并使用示波器或 SignalT ap 逻辑分析仪进行观察调试，如图所示。



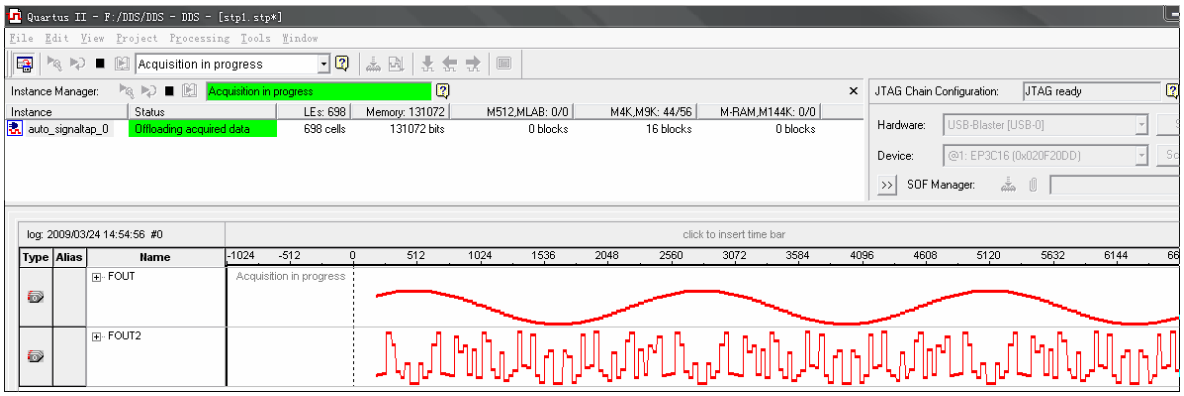
（a）三角波


Quartus II - F:/DDS/DDS - DDS - [stp1.stp*]
File Edit View Project Processing Tools Window
Acquisition in progress
Instance Manager:
Acquisition in progress
Instance Status LEs: 698 Memory: 131072 M512,MLAB: 0/0 M4K,M9K: 44/56 M-RAM,M144K: 0/0
auto_signaltap_0 Offloading acquired data 698 cells 131072 bits 0 blocks 16 blocks 0 blocks
JTAG Chain Configuration: JTAG ready
Hardware: USB-Blaster [USB-0]
Device: @1: EP3C16 (0x020F20DD)
SOF Manager:
log: 2009/03/24 14:54:56 #0
click to insert time bar
Type Alias Name
-1024 -512 0 512 1024 1536 2048 2560 3072 3584 4096 4608 5120 5632
FOUT
FOUT2
Acquisition in progress

（b）锯齿波


Quartus II - F:/DDS/DDS - DDS - [stp1.stp*]
File Edit View Project Processing Tools Window
Acquisition in progress
Instance Manager:
Acquisition in progress
Instance Status LEs: 698 Memory: 131072 M512,MLAB: 0/0 M4K,M9K: 44/56 M-RAM,M144K: 0/0
auto_signaltap_0 Offloading acquired data 698 cells 131072 bits 0 blocks 16 blocks 0 blocks
JTAG Chain Configuration: JTAG ready
Hardware: USB-Blaster [USB-0]
Device: @1: EP3C16 (0x020F20DD)
SOF Manager:
log: 2009/03/24 14:54:56 #0
click to insert time bar
Type Alias Name
-1024 -512 0 512 1024 1536 2048 2560 3072 3584 4096 4608 5120 5632
FOUT
FOUT2
Acquisition in progress

（c）方波


Quartus II - F:/DDS/DDS - DDS - [stp1.stp*]
File Edit View Project Processing Tools Window
Acquisition in progress
Instance Manager:
Acquisition in progress
Instance Status LEs: 698 Memory: 131072 M512,MLAB: 0/0 M4K,M9K: 44/56 M-RAM,M144K: 0/0
auto_signaltap_0 Offloading acquired data 698 cells 131072 bits 0 blocks 16 blocks 0 blocks
JTAG Chain Configuration: JTAG ready
Hardware: USB-Blaster [USB-0]
Device: @1: EP3C16 (0x020F20DD)
SOF Manager:
log: 2009/03/24 14:54:56 #0
click to insert time bar
Type Alias Name
-1024 -512 0 512 1024 1536 2048 2560 3072 3584 4096 4608 5120 5632 6144
FOUT
FOUT2
Acquisition in progress

（d）正弦波

8、试验者自行分析调频，调幅数据，实现对频率以及峰值的精确控制。

六、参考程序

主文件：

```
#include "system.h"
#include "string.h"
#include "altera_avalon_pio_regs.h"
```

```
#include "altera_avalon_uart_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
#include "UART.h"
alt_u8 counter=0;
void delay(alt_u16 temp)
{
 alt_u16 i=0;
 while (i<temp)i++;
}
int main (void) __attribute__ ((weak, alias ("alt_main")));
int alt_main (void)
{
 inti_UART0();
 uart_WRString("system is start!!\r\n");
 printf("system is start!!\r\n");
 while (1)
 {
  IOWR_ALTERA_AVALON_PIO_DATA(FWORD_BASE,
  IORD_ALTERA_AVALON_PIO_DATA(FWR_BASE));
  IOWR_ALTERA_AVALON_PIO_DATA(FRS_BASE,
  IORD_ALTERA_AVALON_PIO_DATA(SELE_BASE));
  delay(1600);
 }
 return 0;
}
```

头文件：

```
#ifndef UART_H_
#define UART_H_
#include "system.h"
#include "string.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_uart_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
alt_u16 contorl=0;
void uart_WRData(unsigned char data )
{ //等待发送寄存器为空
 while(!(IORD_ALTERA_AVALON_UART_STATUS(UART_BASE)&0X20));
 IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE, data);
}
void uart_WRString(unsigned char *s)
{
  while(*s) uart_WRData(*s++);
```

```
 }
static void handle_UART_interrupts(void* context, alt_u32 id)
{
  alt_u16 data;
 /****************在下面开始编写用户中断处理程*************************/
  data =IORD_ALTERA_AVALON_UART_RXDATA(UART_BASE);
  contorl=data;
  IOWR_ALTERA_AVALON_PIO_DATA(VALUE_BASE, contorl);
  uart_WRString("OK!!\r\n");
/*****************************************************************/
 }
static void inti_UART0(void)
{
  alt_u8 count=1;
  alt_irq_init(ALT_IRQ_BASE);
  alt_irq_register( UART_IRQ, (void *)& count, handle_UART_interrupts); //
注册中断函数
  IOWR_ALTERA_AVALON_UART_CONTROL(UART_BASE,0x080); //允许接收完成中断
}
#endif
```

管脚分配：

set_location_assignment PIN_31 -to sys_clk

set_location_assignment PIN_209 -to reset

set_location_assignment PIN_181 -to rxd

set_location_assignment PIN_183 -to txd

set_location_assignment PIN_185 -to da_clk

set_location_assignment PIN_182 -to FWORD[0]

set_location_assignment PIN_184 -to FWORD[1]

set_location_assignment PIN_186 -to FWORD[2]

set_location_assignment PIN_188 -to FWORD[3]

set_location_assignment PIN_194 -to FWORD[4]

set_location_assignment PIN_196 -to FWORD[5]

set_location_assignment PIN_198 -to FWORD[6]

set_location_assignment PIN_200 -to FWORD[7]

set_location_assignment PIN_202 -to FRS[0]

set_location_assignment PIN_207 -to FRS[1]

```
set_location_assignment PIN_216 -to FOUT[0]
set_location_assignment PIN_218 -to FOUT[1]
set_location_assignment PIN_220 -to FOUT[2]
set_location_assignment PIN_222 -to FOUT[3]
set_location_assignment PIN_224 -to FOUT[4]
set_location_assignment PIN_230 -to FOUT[5]
set_location_assignment PIN_232 -to FOUT[6]
set_location_assignment PIN_234 -to FOUT[7]
set_location_assignment PIN_187 -to FOUT2[0]
set_location_assignment PIN_189 -to FOUT2[1]
set_location_assignment PIN_195 -to FOUT2[2]
set_location_assignment PIN_199 -to FOUT2[3]
set_location_assignment PIN_201 -to FOUT2[4]
set_location_assignment PIN_203 -to FOUT2[5]
set_location_assignment PIN_214 -to FOUT2[6]
set_location_assignment PIN_217 -to FOUT2[7]
```

实验三　　高速ADC实验

一、实验目的

进一步熟练使用 SOPC 软件及其配置方法，理解 AD 转换实验原理，掌握其实现方法。

二、实验模块

FPGA 主控制板模块；ADC 模数转换模块。

三、实验原理

本实验采用独立双通道的高速 AD---AD9288，其转换频率可以达到 100MHZ，详见 AD9288.PDF 。

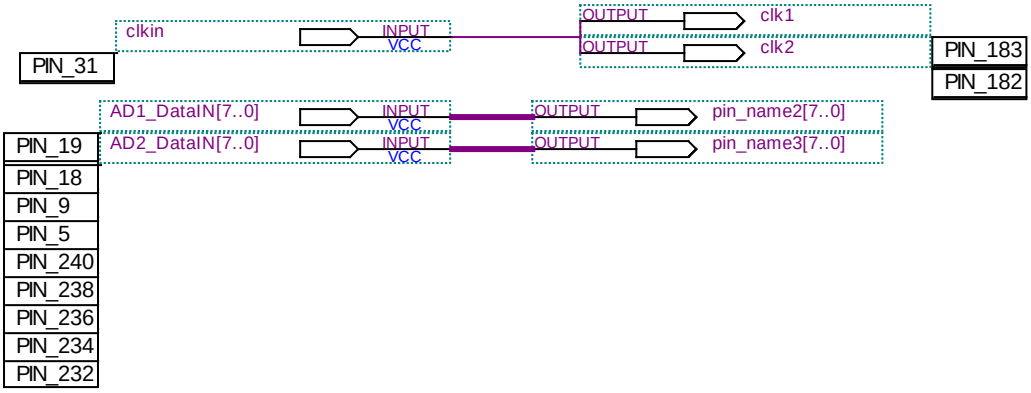
四、实验内容

设置信号发生器输出正弦波、方波、三角波，幅度小于 1v，频率大于 1MHZ，由 AD 采样到 FPGA 中进行波形数据存储分析，并使 SigalTip II Logic Analyzer 来观察输入波形，并判断其频率大小。

五、实验步骤

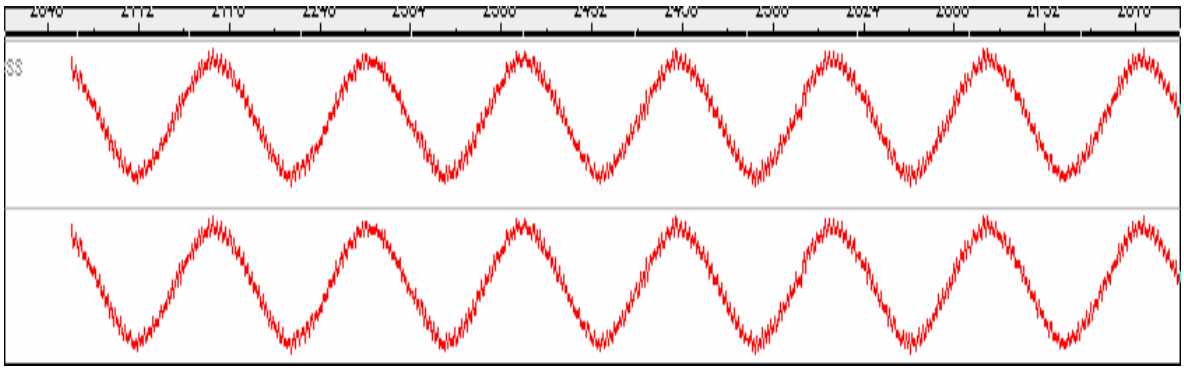
1、利用 Quartus II 软件建立工程命名为 ADC，，实现顶层模块设计。

2、建立 File－New－Block Diagram/Schematic File，加入输入输出，分配管脚，用排线按顺序连接硬件电路，如图所示。

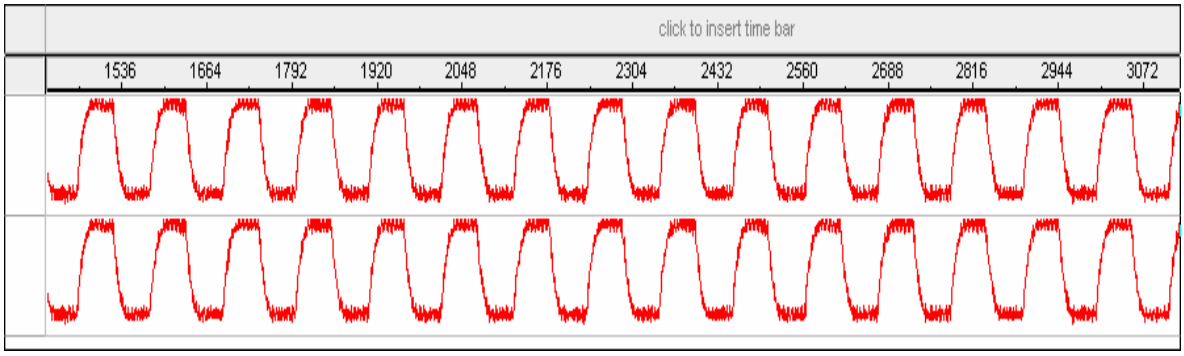


顶层视图

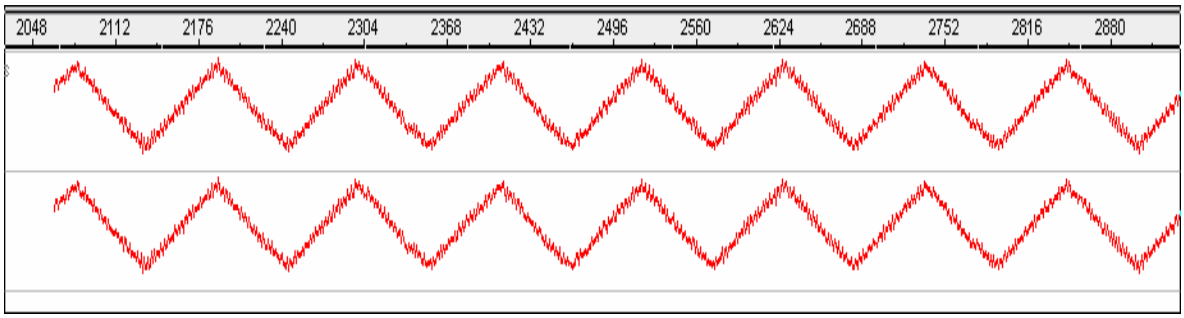
3、下载。首先启动 Quartus II 下的 Programer，把 ADC.sof 文件配置到 FPGA 中。用电缆将信号源产生的模拟信号接入 AD9288 模块，将 AD9288 的数字输出接入 FPGA，通过 SignalT ap 逻辑分析仪观察波形，如图所示。



（a）正弦波



（b）方波



（c）三角波

管脚分配：

```
set_location_assignment PIN_19 -to AD1_DataIN[0]
set_location_assignment PIN_13 -to AD1_DataIN[1]
set_location_assignment PIN_6 -to AD1_DataIN[2]
set_location_assignment PIN_4 -to AD1_DataIN[3]
set_location_assignment PIN_239 -to AD1_DataIN[4]
set_location_assignment PIN_237 -to AD1_DataIN[5]
set_location_assignment PIN_235 -to AD1_DataIN[6]
```

```
set_location_assignment PIN_233 -to AD1_DataIN[7]
set_location_assignment PIN_18 -to AD2_DataIN[0]
set_location_assignment PIN_9 -to AD2_DataIN[1]
set_location_assignment PIN_5 -to AD2_DataIN[2]
set_location_assignment PIN_240 -to AD2_DataIN[3]
set_location_assignment PIN_238 -to AD2_DataIN[4]
set_location_assignment PIN_236 -to AD2_DataIN[5]
set_location_assignment PIN_234 -to AD2_DataIN[6]
set_location_assignment PIN_232 -to AD2_DataIN[7]
set_location_assignment PIN_183 -to clk1
set_location_assignment PIN_182 -to clk2
set_location_assignment PIN_31 -to clkin
```

实验四　静态数码管显示实验

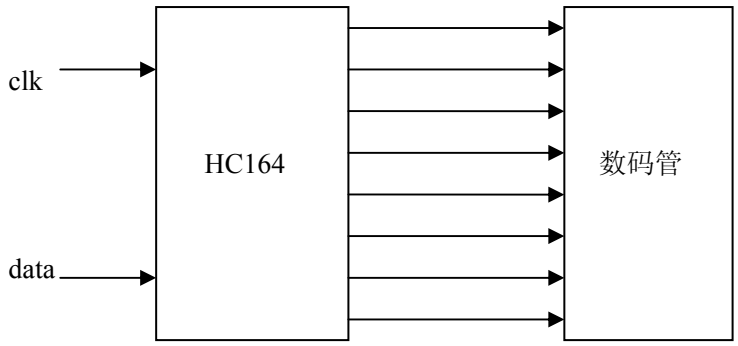
一、 实验目的

1、 掌握数码管静态显示的原理；

2、 学习用 SPI 控制核驱动数码管的原理及编程方法。

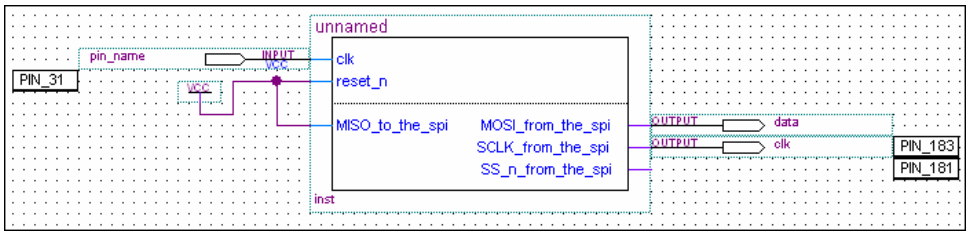
三、实验原理

数码管静态显示是利用 HC164 将数据线送来的数据在时钟下并行送到数码管段口，点亮数码段，显示相应的数据。本实验用 8 个数码管循环显示 0-9 数字，每显示一个数向右移动一位，延时后再显示另一个数。



四、实验步骤

1、打开 quartus 环境，新建工程，命名为 seg8，打开 SOPC Builder，键入 cpu，ram，SPI 控制核，分配管脚后编译下载，如下图所示。



2、打开 nios 环境，编写程序，实现数码管从 0-9 循环显示。

五、参考程序

```
#include "system.h"
#include "string.h"
#include "alt_types.h"
#include "altera_avalon_spi_regs.h"
void delay(unsigned long del)
{
  while(del--);
}
unsigned char spiTransferByte(unsigned char  tempdat)
{
```

```
while(!(IORD_ALTERA_AVALON_SPI_STATUS(SPI_BASE)&ALTERA_AVALON_SPI_STATUS_
TRDY_MSK));
        IOWR_ALTERA_AVALON_SPI_TXDATA(SPI_BASE,tempdat);
while(!(IORD_ALTERA_AVALON_SPI_STATUS(SPI_BASE)&ALTERA_AVALON_SPI_STATUS_
TMT_MSK));
while(!(IORD_ALTERA_AVALON_SPI_STATUS(SPI_BASE)&ALTERA_AVALON_SPI_STATUS_
RRDY_MSK));
  return IORD_ALTERA_AVALON_SPI_RXDATA(SPI_BASE);
}
int main()
{
 alt_u8 i=0;
 alt_u8
SEGBuufer[11]={0x11,0x7D,0X32,0X92,0XD4,0X98,0X18,0XD3,0X10,0X90,0XFF};
while (1)
  {spiTransferByte(SEGBuufer[i]);
    i++;
    if(i>9)
    i=0;
   delay(8900000);
   delay(8900000);
   delay(8900000);
   delay(5900000);
  }
  return 0;
}
```

管脚分配：

set_location_assignment PIN_31 -to clk

set_location_assignment PIN_181 -to seg_clk

set_location_assignment PIN_183 -to seg_dat

实验五　VGA彩条显示实验

一、实验目的

进一步熟练使用 SOPC 软件及其配置方法，理解 VGA 电脑显示器显示原理，掌握其时序描述方法。

二、实验模块

VGA 控制器接口模。

三、实验原理

VGA 显示控制图像信号通过电缆传输到显示器上并显示出来，要包括两种：CRT（Cathode Ray Tube，阴极射线管）和 LCD（Liquid Crystal Display，液晶显示屏）。CRT 通过帧同步信号和行同步信号控制电子枪的电子束逐行逐点地扫描，将电子打在荧光点上，使之发光。通过视觉暂留作用，看到的就是一幅完整的画面。LCD 与 CRT 类似，也是动态的扫描。但 CRT 是模拟方式的，通过电路控制，电子束可以任意移动；而 LCD 是数字方式的，只有位置固定的电流通路，所以只能通过电路矩阵逐行扫描，而不能逐点，即一行上的所有的点同时工作。本模块以高速 DA 转换芯片 ADV7120 实现 VGA 对其同轴电缆的驱动。详细见 ADV7120.pdf 。

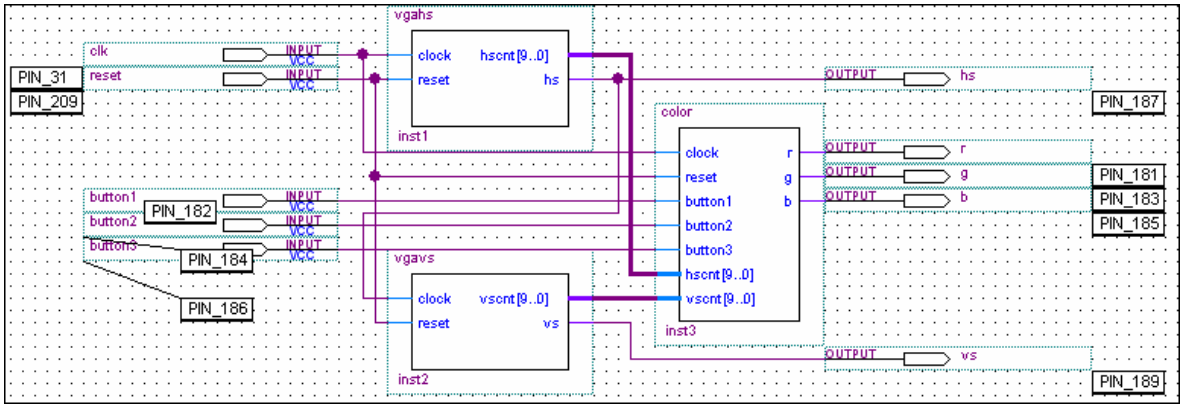
四、实验内容

编写程序实现 VGA 彩条显示，像素 640x480，刷新频率 60Hz，实现 8 位色的彩条显示。

五、实验步骤

1、利用 Quartus II 软件建立工程命名为 VGA,并选择 cycloneIII 器件。新建并编写 VHDL 程序。然后生成框图文件。

2、建立 File－New－Block Diagram/Schematic File,加入生成的框图文件与输入输出 I/O 口，并在重命名后分配管脚，用排线按顺序连接硬件电路。编译后下载到 CFPGA 中。



六、参考程序

颜色显示模块程序：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity color is
    Port (  clock : in std_logic;
            reset : in std_logic;
            button1:in std_logic;
            button2:in std_logic;
            button3:in std_logic;
            hscnt  : in std_logic_vector(9 downto 0);
            vscnt  : in std_logic_vector(9 downto 0);
            r  : out std_logic;
            g  : out std_logic;
            b  : out std_logic);
end color;
architecture Behavioral of color is
signal  vs_num: integer:=0;
signal  hs_num: integer:=0;
signal  qvs: integer:=460;
signal  qhs: integer:=620;
signal  rgb:std_logic_vector(2 downto 0);
      begin
          c:PROCESS(CLOCK)
            BEGIN
            if reset='1' then
              IF CLOCK'EVENT AND CLOCK='1' THEN
                IF((hscnt<640)and(vscnt<480)) then
                  if (button1='0')then
                          if((hscnt>=0)and(hscnt<80)and(vscnt>=0)and(vscnt<=480))then
```

```
            rgb<="100";
        elsif((hscnt>=80)and(hscnt<160)and(vscnt>=0)and(vscnt<=480))then
            rgb<="010";
        elsif((hscnt>=160)and(hscnt<240)and(vscnt>=0)and(vscnt<=480))then
            rgb<="001";
        elsif((hscnt>=240)and(hscnt<320)and(vscnt>=0)and(vscnt<=480))then
            rgb<="000";
        elsif((hscnt>=320)and(hscnt<400)and(vscnt>=0)and(vscnt<=480))then
            rgb<="111";
        elsif((hscnt>=400)and(hscnt<480)and(vscnt>=0)and(vscnt<=480))then
            rgb<="110";
        elsif((hscnt>=480)and(hscnt<560)and(vscnt>=0)and(vscnt<=480))then
            rgb<="011";
        elsif((hscnt>=560)and(hscnt<640)and(vscnt>=0)and(vscnt<=480))then
            rgb<="101";
      else
          rgb<="001";
      end if;
  elsif (button2='0')then
      if((hscnt>=0)and(hscnt<80)and(vscnt>=0)and(vscnt<=480))then
            rgb<="101";
        elsif((hscnt>=80)and(hscnt<160)and(vscnt>=0)and(vscnt<=480))then
            rgb<="011";
        elsif((hscnt>=160)and(hscnt<240)and(vscnt>=0)and(vscnt<=480))then
            rgb<="110";
        elsif((hscnt>=240)and(hscnt<320)and(vscnt>=0)and(vscnt<=480))then
            rgb<="111";
        elsif((hscnt>=320)and(hscnt<400)and(vscnt>=0)and(vscnt<=480))then
            rgb<="000";
        elsif((hscnt>=400)and(hscnt<480)and(vscnt>=0)and(vscnt<=480))then
```

```
                rgb<="001";
            elsif((hscnt>=480)and(hscnt<560)and(vscnt>=0)and(vscnt<=480))then
                rgb<="010";
            elsif((hscnt>=560)and(hscnt<640)and(vscnt>=0)and(vscnt<=480))then
                rgb<="100";
            else
                rgb<="001";
            end if;
        elsif (button3='0')then
            if((hscnt>=0)and(hscnt<80)and(vscnt>=0)and(vscnt<=480))then
                rgb<="100";
            elsif((hscnt>=80)and(hscnt<160)and(vscnt>=0)and(vscnt<=480))then
                rgb<="010";
            elsif((hscnt>=160)and(hscnt<240)and(vscnt>=0)and(vscnt<=480))then
                rgb<="001";
            elsif((hscnt>=240)and(hscnt<320)and(vscnt>=0)and(vscnt<=480))then
                rgb<="111";
            elsif((hscnt>=320)and(hscnt<400)and(vscnt>=0)and(vscnt<=480))then
                rgb<="000";
            elsif((hscnt>=400)and(hscnt<480)and(vscnt>=0)and(vscnt<=480))then
                rgb<="001";
            elsif((hscnt>=480)and(hscnt<560)and(vscnt>=0)and(vscnt<=480))then
                rgb<="010";
            elsif((hscnt>=560)and(hscnt<640)and(vscnt>=0)and(vscnt<=480))then
                rgb<="100";
            else
                rgb<="001";
            end if;
        else
            if((vscnt=200)and(hscnt>280)and(hscnt<360))then
```

```
                rgb<="100";
            elsif((vscnt=280)and(hscnt>280)and(hscnt<360))then
                rgb<="100";
            elsif((hscnt=280)and(vscnt>200)and(vscnt<280))then
                rgb<="100";
            elsif((hscnt=360)and(vscnt>200)and(vscnt<280))then
                rgb<="100";
            elsif(vscnt=qvs or hscnt=qhs)then
                vs_num<=vs_num+1;
                rgb<="110";
              if(vscnt>=160 and vscnt<=480)then
                 rgb<="100";
              end if;
              if(hscnt>=160 and hscnt<=480)then
                 rgb<="010";
              end if;
              if(vs_num>=3080) then vs_num<=0;
                 if(qvs>0)then qvs<=qvs-1;else qvs<=460; end if;
                 if(qhs>0)then qhs<=qhs-1;else qhs<=620; end if;
              end if;
            else
                rgb<="001";
            end if;
          end if;
       ELSE
        rgb<="000";   -- 消隐信号--黑色
       END IF;
    END IF;
  end if;
END PROCESS;
```

```
            r<=rgb(2);
            g<=rgb(1);
            b<=rgb(0);
end Behavioral;
```

管脚分配：

```
set_location_assignment PIN_181 -to r
set_location_assignment PIN_183 -to g
set_location_assignment PIN_185 -to b
set_location_assignment PIN_189 -to hs
set_location_assignment PIN_187 -to vs
set_location_assignment PIN_209 -to reset
set_location_assignment PIN_31 -to clk
set_location_assignment PIN_182 -to button1
set_location_assignment PIN_184 -to button2
set_location_assignment PIN_186 -to button3
```

实验六　PS2 键盘实验

一、实验目的

进一步熟练使用 SOPC 软件及其配置方法，理解 PS2 键盘编码原理，掌握其读键盘码方法。

二、实验模块

FPGA 主控制板模块；PS2 控制器接口模块；串口通信模块。

三、实验原理

键盘的处理器花费很多的时间来扫描或监视按键矩阵，如果它发现有键被按下释放或按住键盘，将发送扫描码的信息包到计算机。扫描码有两种不同的类型：通码和断码，当一个键被按下或按住就发送通码，当一个键被释放就发送断码。每个按键被分配了唯一的通码和断码，这样主机通过查找唯一的扫描码就可以测定是哪个按键。每个键一整套的通断码组成了扫描码集。有三套标准的扫描码集：分别是第一套，第二套和第三套。所有现代的键盘默认使用第二套扫描码；详细请参考有关技术资料。

四、实验内容

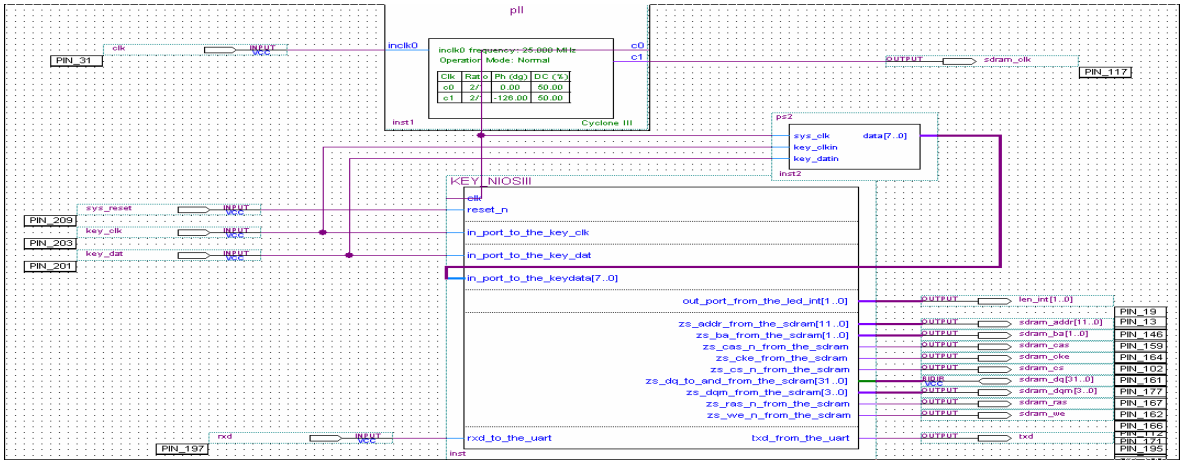
把键盘数据通过串口发送到电脑主机上显示哪个键被按下。

五、实验步骤

1、利用 Quartus II 软件建立工程命名为 PS2，建立键盘通码的采集模块文件。

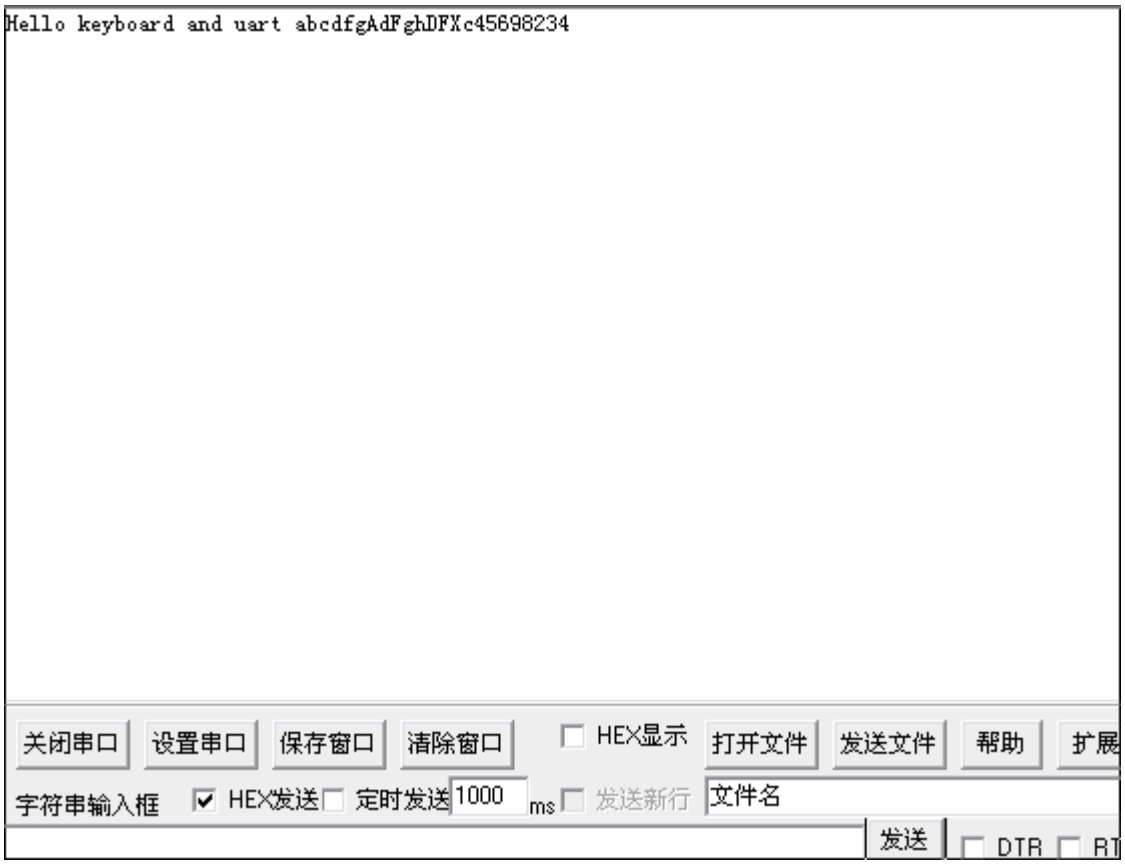
2、启动 SOPC Builder 建立 CPU 命名 key_nios2；为 CPU 添加 Nios II 软核，选择 Nios II/S, 建立片内 15K 的 RAM，并为 CPU 添加复位地址为 RAM 地址。添加 UART 模块设置波特率为 115200；8 位数据位，无校验位，1 停止位；添加 SDRAM，并进行相关设置；然后生成 CPU。

3、建立 File－New－Block Diagram/Schematic File，加入生成的框图文件与输入输出 I/O 口，并在重命名后分配管脚，用排线按顺序连接硬件电路，如下图所示。



顶层视图

4、编译后启动 Quartus II 下的 Programer，把 PS2.sof 文件配置到 FPGA 中，然后在 Nios II IDE 中 Run as---Nios II Hardware.按下按键会观察到电脑串口发出的键盘数据，如下图所示。



六、参考程序

本例程为键盘通码的采集程序，数据采集后再由 Nios II 核进行处理判断。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ps2 is
port
   (
      sys_clk:in std_logic;
      key_clkin : in std_logic;
      key_datin:in std_logic;
      data:out std_logic_vector(7 downto 0)
    );
 end entity ps2;
 architecture one of ps2 is
 signal clk:std_logic;
 signal num : std_logic_vector(4 downto 0);
```

```
 signal m: std_logic_vector( 9 downto 0 );
 begin
A: process(sys_clk)---read keybarod's clk in
   begin
      if sys_clk'event and sys_clk = '1' then
           if key_clkin = '0' then
              clk <= key_clkin;
            else
              clk <= '1';
         end if;
      end if;
   end process;
B:    process(key_clkin,clk,m)    ---based on clk read then dat bit
     begin
      if clk'event and clk = '0' then
            m <= key_datin & m( 9 downto 1 );
        end if;
   end process;
C: process(clk,key_datin,m)
   begin
      if clk'event and clk = '1' then
           data<=m(7 downto 0);
      end if;
     end process;
 end architecture;
```

管脚分配：

```
    set_location_assignment PIN_209 -to sys_reset
    set_location_assignment PIN_146 -to sdram_addr[0]
    set_location_assignment PIN_145 -to sdram_addr[1]
    set_location_assignment PIN_144 -to sdram_addr[2]
    set_location_assignment PIN_111 -to sdram_addr[3]
    set_location_assignment PIN_110 -to sdram_addr[4]
    set_location_assignment PIN_109 -to sdram_addr[5]
    set_location_assignment PIN_108 -to sdram_addr[6]
    set_location_assignment PIN_107 -to sdram_addr[7]
    set_location_assignment PIN_106 -to sdram_addr[8]
    set_location_assignment PIN_103 -to sdram_addr[9]
```

```
set_location_assignment PIN_147 -to sdram_addr[10]
set_location_assignment PIN_160 -to sdram_addr[11]
set_location_assignment PIN_159 -to sdram_ba[0]
set_location_assignment PIN_148 -to sdram_ba[1]
set_location_assignment PIN_164 -to sdram_cas
set_location_assignment PIN_102 -to sdram_cke
set_location_assignment PIN_117 -to sdram_clk
set_location_assignment PIN_161 -to sdram_cs
set_location_assignment PIN_177 -to sdram_dq[0]
set_location_assignment PIN_176 -to sdram_dq[1]
set_location_assignment PIN_175 -to sdram_dq[2]
set_location_assignment PIN_174 -to sdram_dq[3]
set_location_assignment PIN_173 -to sdram_dq[4]
set_location_assignment PIN_171 -to sdram_dq[5]
set_location_assignment PIN_169 -to sdram_dq[6]
set_location_assignment PIN_168 -to sdram_dq[7]
set_location_assignment PIN_100 -to sdram_dq[8]
set_location_assignment PIN_99 -to sdram_dq[9]
set_location_assignment PIN_98 -to sdram_dq[10]
set_location_assignment PIN_95 -to sdram_dq[11]
set_location_assignment PIN_94 -to sdram_dq[12]
set_location_assignment PIN_93 -to sdram_dq[13]
set_location_assignment PIN_88 -to sdram_dq[14]
set_location_assignment PIN_87 -to sdram_dq[15]
set_location_assignment PIN_142 -to sdram_dq[16]
set_location_assignment PIN_139 -to sdram_dq[17]
set_location_assignment PIN_137 -to sdram_dq[18]
set_location_assignment PIN_135 -to sdram_dq[19]
set_location_assignment PIN_134 -to sdram_dq[20]
set_location_assignment PIN_133 -to sdram_dq[21]
set_location_assignment PIN_132 -to sdram_dq[22]
set_location_assignment PIN_131 -to sdram_dq[23]
set_location_assignment PIN_128 -to sdram_dq[24]
set_location_assignment PIN_127 -to sdram_dq[25]
set_location_assignment PIN_126 -to sdram_dq[26]
set_location_assignment PIN_120 -to sdram_dq[27]
set_location_assignment PIN_119 -to sdram_dq[28]
set_location_assignment PIN_118 -to sdram_dq[29]
```

```
set_location_assignment PIN_114 -to sdram_dq[30]
set_location_assignment PIN_113 -to sdram_dq[31]
set_location_assignment PIN_167 -to sdram_dqm[0]
set_location_assignment PIN_101 -to sdram_dqm[1]
set_location_assignment PIN_143 -to sdram_dqm[2]
set_location_assignment PIN_112 -to sdram_dqm[3]
set_location_assignment PIN_166 -to sdram_we
set_location_assignment PIN_31 -to clk
set_location_assignment PIN_203 -to key_clk
set_location_assignment PIN_201 -to key_dat
set_location_assignment PIN_19 -to len_int[1]
set_location_assignment PIN_13 -to len_int[0]
set_location_assignment PIN_197 -to rxd
set_location_assignment PIN_195 -to txd
set_location_assignment PIN_162 -to sdram_ras
```

实验七　USB数据读写实验

一、实验目的

进一步熟练使用 SOPC 软件及其配置方法，理解 USB 数据传输原理，掌握其时序描述和读写方法。

二、实验模块

USB 控制器接口模块；FPGA 主控制板模块。

三、实验原理

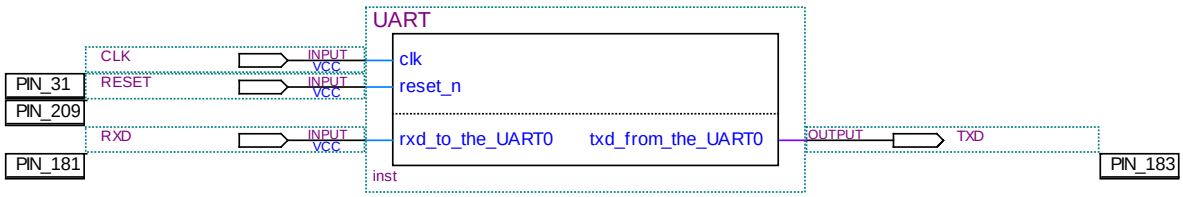
USB(Universal Serial Bus)是 1995 年 Microsoft、Compaq、IBM 等公司联合制定的一种新的 PC 串行通信协议。FT232BM 的主要功能是进行 USB 和串口之间的协议转换。芯片一方面可从主机接收 USB 数据，并将其转换为串口的数据流格式发送给外设；另一方面外设可通过虚拟串口将数据转换为 USB 的数据格式传回主机。中间的转换工作全部由芯片自动完成，开发者无须考虑固件的设计。详细见 FT232BM.pdf 。

四、实验内容

安装 USB 串口驱动程序，PC 通过 USB 口把数据传到 FPGA，再由 FPGA 通过 USB 口传回到 PC 机。

五、实验步骤

1、利用 Quartus II 软件建立工程命名为 USB_WR,并选择 Cyclone III 器件 EP3C16Q240C8。启动 SOPC builder，添加 CPU 命名为 NIOS2。为 CPU 添加 Nios II 软核，选择 Nios II/S 建立片内 10K 的 on-chip ram 命名为 RAM，并为 CPU 添加复位地址为 RAM 地址。然后生成 CPU；分配管脚，并编译文件。



2、根据 FT232 的时序在 NIOS II IDE 中编写应用程序，并观察 USB 和 PC 机的通信情况。

3、实验程序　参考 UART 读写实验例程。

管脚分配：

set_location_assignment PIN_31 -to CLK

set_location_assignment PIN_209 -to RESET

```
set_location_assignment PIN_181 -to RXD
set_location_assignment PIN_183 -to TXD
```

实验八 TFT真彩屏实验

一、实验目的

1、 进一步掌握 TFT 彩色液晶显示；

2、了解触摸屏原理，掌握其简单变成。

二、实验模块

TFT 液晶模块、TFT 控制器接口模块；FPGA 主控制板模块。

三、实验原理

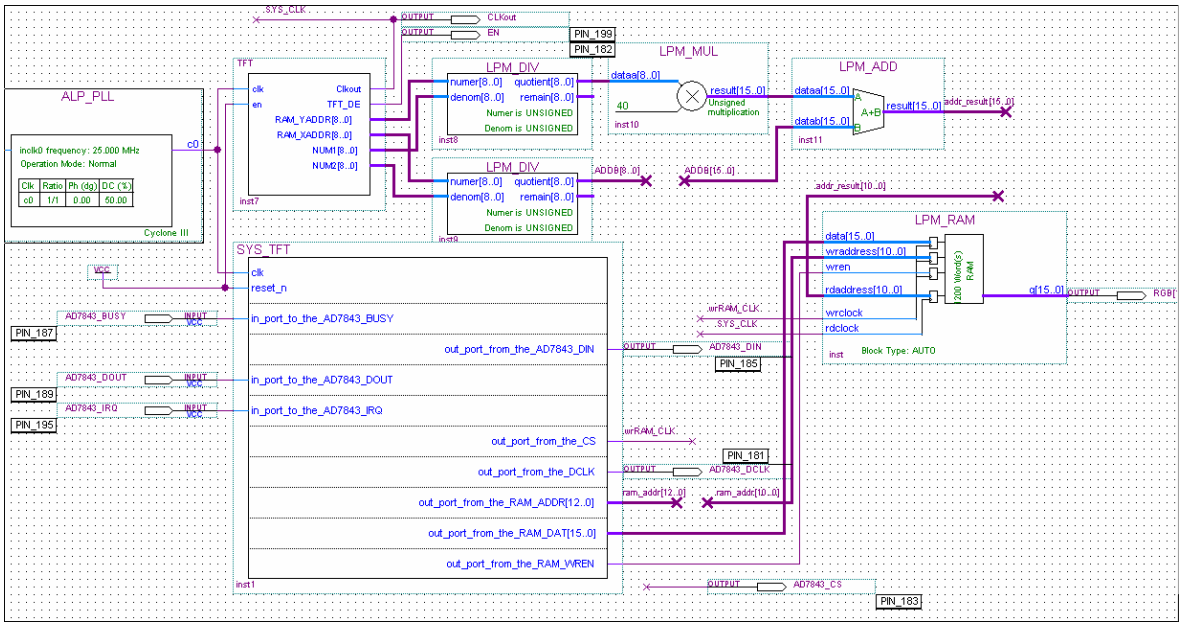
触摸屏由两部分构成即液晶显示部分和触摸检测部分二者相互独立。本实验使用的控制芯片是 ADI 公司生产的 AD7843 四线式触摸屏控制器，目前该控制器已广泛应用于电阻式触摸屏输入系统中。AD7843 数字转换器在一个 12 位逐次逼近式比较寄存器 SAR ADC 架构上集成了用于驱动触摸屏的低通阻抗开关。其测量原理可以参考 AD7843.pdf 和张井刚的《AD7843 在触摸屏系统中的应用》。

四、实验步骤

1、新建 Quartues II 工程文件，然后按照真彩屏的显示时序，设置显示字符及按键。

2、启动 SOPC Builder 然后，添加 CPU，添加 ram，添加 AD7843 所需要的控制及数据 PIO 端口、JTAG 等组件，并重命名，生成 CPU。

3、建立 block 文件，搭建硬件图，如下图所示。编译，下载。



4、启动 NiosII ide 并建立空白工程，然后编写程序实现汉字、字符、按钮的显示效果。

5、参考 ad7843.pdf，完成 AD7843 的初始化时序工作，然后分别从 x，y 两个方向读出其 ADC 转换的电压值，然后再在每个按钮上多次触摸，确定该按钮在受到触摸时的输出电压范围。并根

据此范围写出控制程序。编写程序实现“济南”二字的颜色变化及 0—99 的加减变化。

五、参考程序

主文件：

```
#include "system.h"
#include "string.h"
#include "stdio.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
#include "ad7843.h"
#include "RAM.h"
#include "pio_irq.h"
extern void WRsre_NUM(unsigned int num);
int main (void) __attribute__ ((weak, alias ("alt_main")));
int alt_main (void)
{ unsigned int i=0;
  alt_irq_init(ALT_IRQ_BASE);
  Init_AD7843_IRQExt_interrupt();
  ReadAD7843_X() ;
  ReadAD7843_Y() ;
  clr_screen();
  WR_HZ12(0,0,0,frontcolor);//济
  WR_HZ12(0,13,1,frontcolor);//南
  printf("system is start!!\r\n");
  WR_BUttoncolor(3,30,10,color1);
  WR_BUttoncolor(9,30,10,color2);
  WR_BUttoncolor(15,30,10,color3);
  WR_BUtton(23,22,5);
  WR_BUtton(23,30,0);
  WRsre_NUM(num);
  while(1)
  {
   ;;
  }
}
```

管脚分配;

set_location_assignment PIN_195 -to AD7843_DCLK

set_location_assignment PIN_189 -to AD7843_CS

set_location_assignment PIN_187 -to AD7843_DIN

```
set_location_assignment PIN_185 -to AD7843_BUSY
set_location_assignment PIN_183 -to AD7843_DOUT
set_location_assignment PIN_181 -to AD7843_IRQ
set_location_assignment PIN_31 -to CLK
set_location_assignment PIN_234 -to EN
set_location_assignment PIN_186 -to RGB[14]
set_location_assignment PIN_188 -to RGB[12]
set_location_assignment PIN_194 -to RGB[10]
set_location_assignment PIN_196 -to RGB[8]
set_location_assignment PIN_198 -to RGB[6]
set_location_assignment PIN_200 -to RGB[4]
set_location_assignment PIN_202 -to RGB[2]
set_location_assignment PIN_207 -to RGB[0]
set_location_assignment PIN_235 -to CLKout
set_location_assignment PIN_201 -to RGB[15]
set_location_assignment PIN_203 -to RGB[13]
set_location_assignment PIN_214 -to RGB[11]
set_location_assignment PIN_217 -to RGB[9]
set_location_assignment PIN_219 -to RGB[7]
set_location_assignment PIN_221 -to RGB[5]
set_location_assignment PIN_223 -to RGB[3]
set_location_assignment PIN_226 -to RGB[1]
```

实验九 SD卡实验

一、实验目的

进一步熟练使用 SOPC 软件及其配置方法，理解 SD 卡原理，掌握其时序描述和读写方法。

二、实验模块

SD 卡实验模块、RS232 串口模块、FPGA 主控制板模块。

三、实验原理

SD 有两个可以选择的通信协议：SD 模式和 SPI 模式。为了电路的简化，本模块实验采用了 SPI 模式。由于 SD 卡的默认模式为 SD 模式，要进入 SPI 模式时，只要在接收到 RESET 命令时，将 CS 拉低即可。关于 SD 卡的模式命令格式详见“SD 卡要点说明”。

四、实验内容

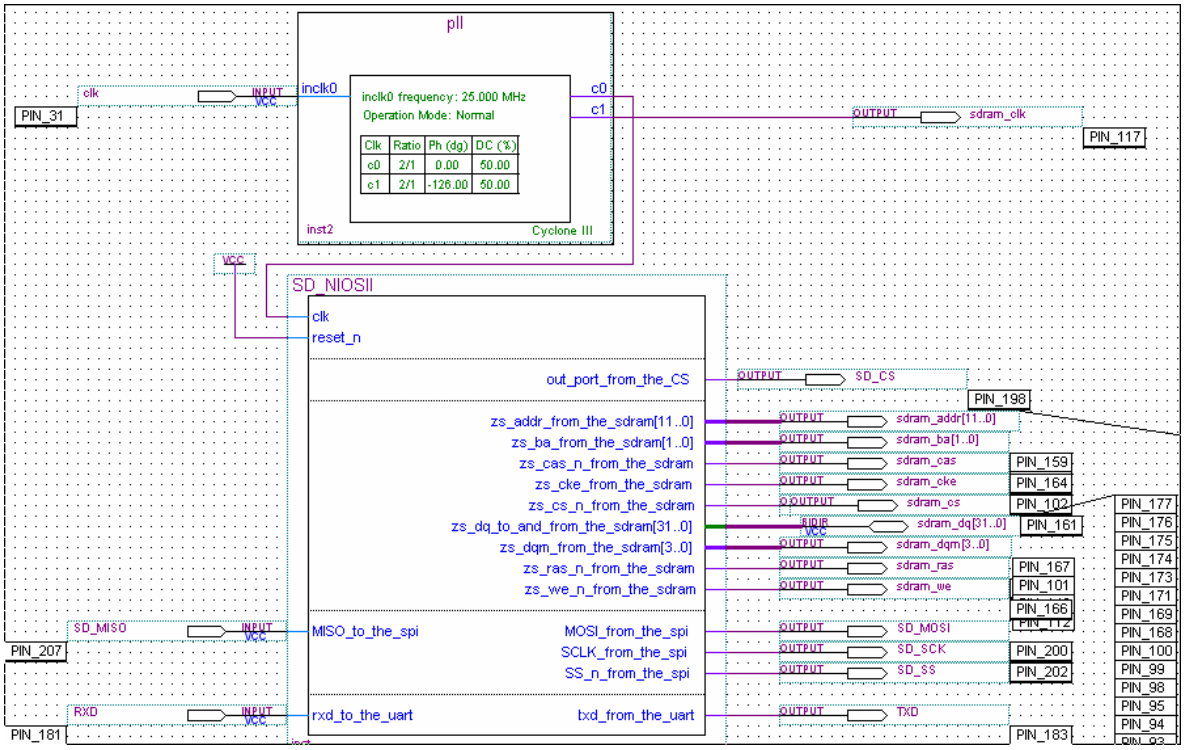
在 SD 卡中存入*.txt 文件，编写程序读出其中文件名极其格式。

五、实验步骤

1、利用 Quartus II 软件建立工程命名为 SD，并选择 Cyclone III 器件 EP3C16Q240C8。

2、建立 SOPC Builder，命名为 Nios2_SD，加入 I/O 口和 UART 串口，建立 15k 的 RAM 空间或使用外部 SDRAM 和 FLASH，加入内部定时器 timer，加入系统 ID，加入 SPI 控制器核，并生成 CPU。

3、新建原理图文件，添加分配管脚，并编译文件。如图所示。



4、启动 Nios II IDE 并重新设置工作目录路径，建立新的 C/C++应用程序；选择 Nios2_SD

的 CPU，和示例“hello word”，生成工程。生成工程文件后，设置系统属性。通过 Project→properties 在弹出对话框中选择 System Library Properties 设置。

5、下载。编译后首先把 Quartus II 中的*.sof 文件下载到 FPGA。然后再由 Nios II IDE 中 Run as---Nios II Hardware.，运行程序。

六、实验程序

主文件：

```
#include <stdio.h>
#include "includes.h"
#include "system.h"
#include "string.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#include "altera_avalon_spi_regs.h"
#include "common.h"
#include "hal.h"
#include "sdinf.h"
#include "fat16.h"
/* Definition of Task Stacks */
#define   TASK_STACKSIZE      2048
OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];
/* Definition of Task Priorities */
#define TASK1_PRIORITY     1
#define TASK2_PRIORITY     2
/* Prints "Hello World" and sleeps for three seconds */
void task1(void* pdata)
{ int i;
   OSTimeDlyHMSM(0, 0, 3, 0);
  if(SdReset()==0)
  {
     printf("Hello sucess\n");
     if(InitFat16()==SD_SUCC){  printf("init sucess\n");
       ReadFileID(ReadRoootDir());
    }
    else printf("inti fail\n");
 }
  else
     printf("SdReset failed\r\n");
  while (1)
  {
    OSTimeDlyHMSM(0, 0, 0, 1);
```

```
    OSTimeDlyHMSM(0, 0, 1, 1);
  }
}
static alt_u8 fun;
/* Prints "Hello World" and sleeps for three seconds */
void task2(void* pdata)
{  while (1)
  {
    scanf("%s",&fun);
    if(fun=='1')
    ReadFileID(ReadRoootDir());
    else if(fun<58)
ReadFileNr(fun-48,TempFilePBP[fun-48].StartClusPosit,TempFilePBP[fun-48].
Size);
    else
    printf("Hello from task2 fun=%d\n",fun);
    break ;
    OSTimeDlyHMSM(0, 0, 1, 0);
  }
}
/* The main function creates two task and starts multi-tasking */
int main(void)
{
  OSTaskCreateExt(task1,
                  NULL,
                  (void *)&task1_stk[TASK_STACKSIZE],
                  TASK1_PRIORITY,
                  TASK1_PRIORITY,
                  task1_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);
  OSTaskCreateExt(task2,
                  NULL,
                  (void *)&task2_stk[TASK_STACKSIZE],
                  TASK2_PRIORITY,
                  TASK2_PRIORITY,
                  task2_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);
  OSStart();
  return 0;
}
```

管脚分配：

```
set_location_assignment PIN_209 -to sys_reset
set_location_assignment PIN_146 -to sdram_addr[0]
set_location_assignment PIN_145 -to sdram_addr[1]
set_location_assignment PIN_144 -to sdram_addr[2]
set_location_assignment PIN_111 -to sdram_addr[3]
set_location_assignment PIN_110 -to sdram_addr[4]
set_location_assignment PIN_109 -to sdram_addr[5]
set_location_assignment PIN_108 -to sdram_addr[6]
set_location_assignment PIN_107 -to sdram_addr[7]
set_location_assignment PIN_106 -to sdram_addr[8]
set_location_assignment PIN_103 -to sdram_addr[9]
set_location_assignment PIN_147 -to sdram_addr[10]
set_location_assignment PIN_160 -to sdram_addr[11]
set_location_assignment PIN_159 -to sdram_ba[0]
set_location_assignment PIN_148 -to sdram_ba[1]
set_location_assignment PIN_164 -to sdram_cas
set_location_assignment PIN_102 -to sdram_cke
set_location_assignment PIN_117 -to sdram_clk
set_location_assignment PIN_161 -to sdram_cs
set_location_assignment PIN_177 -to sdram_dq[0]
set_location_assignment PIN_176 -to sdram_dq[1]
set_location_assignment PIN_175 -to sdram_dq[2]
set_location_assignment PIN_174 -to sdram_dq[3]
set_location_assignment PIN_173 -to sdram_dq[4]
set_location_assignment PIN_171 -to sdram_dq[5]
set_location_assignment PIN_169 -to sdram_dq[6]
set_location_assignment PIN_168 -to sdram_dq[7]
set_location_assignment PIN_100 -to sdram_dq[8]
set_location_assignment PIN_99 -to sdram_dq[9]
set_location_assignment PIN_98 -to sdram_dq[10]
set_location_assignment PIN_95 -to sdram_dq[11]
set_location_assignment PIN_94 -to sdram_dq[12]
set_location_assignment PIN_93 -to sdram_dq[13]
set_location_assignment PIN_88 -to sdram_dq[14]
set_location_assignment PIN_87 -to sdram_dq[15]
set_location_assignment PIN_142 -to sdram_dq[16]
set_location_assignment PIN_139 -to sdram_dq[17]
set_location_assignment PIN_137 -to sdram_dq[18]
set_location_assignment PIN_135 -to sdram_dq[19]
set_location_assignment PIN_134 -to sdram_dq[20]
set_location_assignment PIN_133 -to sdram_dq[21]
set_location_assignment PIN_132 -to sdram_dq[22]
set_location_assignment PIN_131 -to sdram_dq[23]
```

```
set_location_assignment PIN_128 -to sdram_dq[24]
set_location_assignment PIN_127 -to sdram_dq[25]
set_location_assignment PIN_126 -to sdram_dq[26]
set_location_assignment PIN_120 -to sdram_dq[27]
set_location_assignment PIN_119 -to sdram_dq[28]
set_location_assignment PIN_118 -to sdram_dq[29]
set_location_assignment PIN_114 -to sdram_dq[30]
set_location_assignment PIN_113 -to sdram_dq[31]
set_location_assignment PIN_167 -to sdram_dqm[0]
set_location_assignment PIN_101 -to sdram_dqm[1]
set_location_assignment PIN_143 -to sdram_dqm[2]
set_location_assignment PIN_162 -to sdram_ras
set_location_assignment PIN_112 -to sdram_dqm[3]
set_location_assignment PIN_166 -to sdram_we
set_location_assignment PIN_31 -to clk
set_location_assignment PIN_181 -to RXD
set_location_assignment PIN_198 -to SD_CS
set_location_assignment PIN_200 -to SD_MOSI
set_location_assignment PIN_202 -to SD_SCK
set_location_assignment PIN_207 -to SD_MISO
set_location_assignment PIN_183 -to TXD
```

实验十　UC\OS-II操作系统移植实验

一、实验目的

进一步熟练使用 SOPC 软件及其配置方法，理解操作系统工作原理，掌握其在 NIOS II IDE 中的简单使用。

二、实验模块

RS232 串口模块、FPGA 主控制板模块。

三、实验原理

在 Nios II IDE 中使用 MicroC/OS-II RTOS ，开发人员可以根据具体实验任务的需要通过在面板选项中的设定来自己剪裁程序，可以通过 RTOS Option 选项来自己定义所需要的功能选项。

四、实验内容

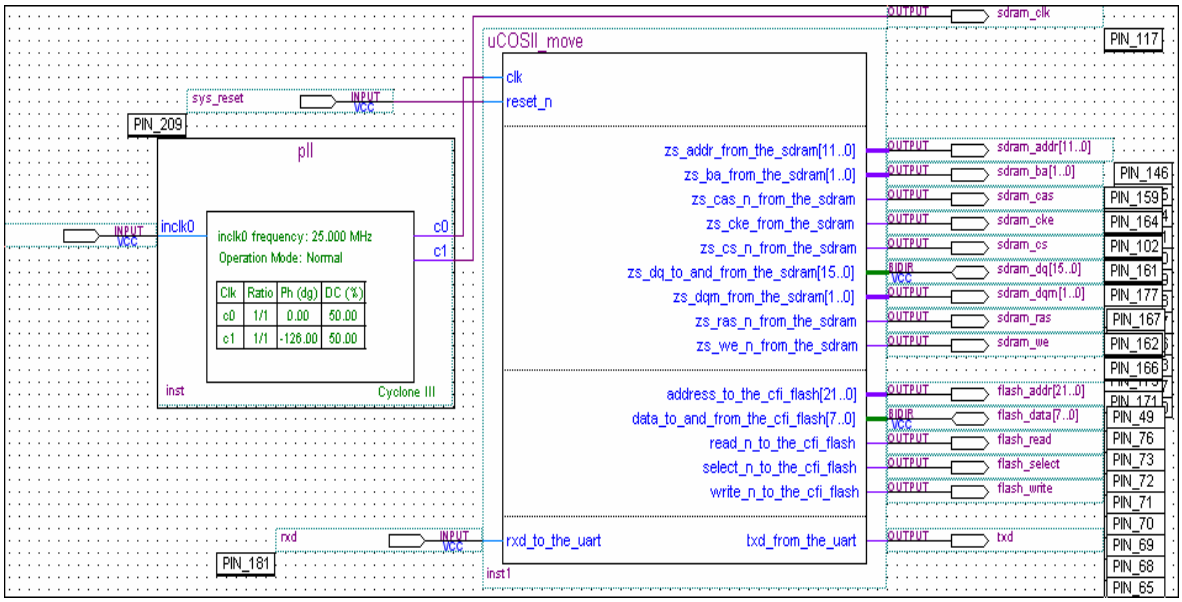
以 Nios II IDE 中软件已经经过剪切的例程 Hello MicroC/OS-II 为例，来进行串口实验。

五、实验步骤

1、利用 Quartus II 软件建立工程命名为 uCOSII,并选择 Cyclone III 器件 EP3C16Q240C8。

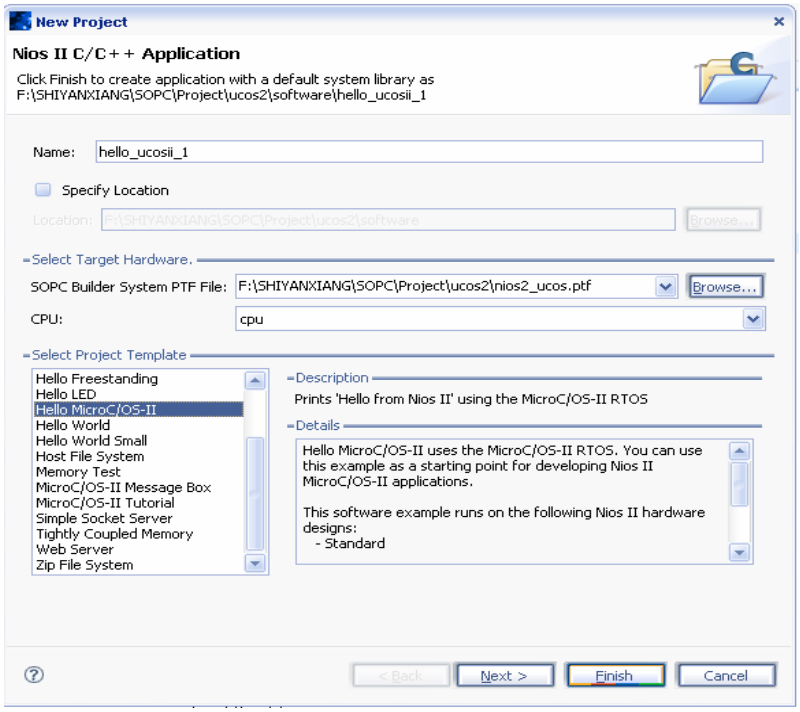
2、建立 SOPC Builder，命名为 Ucosii_move，加入 JTAG_UART 和 UART 串口，建立 15k 的 RAM ，加入外部 SDRAM 和 FLASH，加入内部定时器 timer，加入系统 ID，加入三态桥，并生成 CPU。

3、新建原理图文件，添加分配管脚，并编译文件。如图所示。

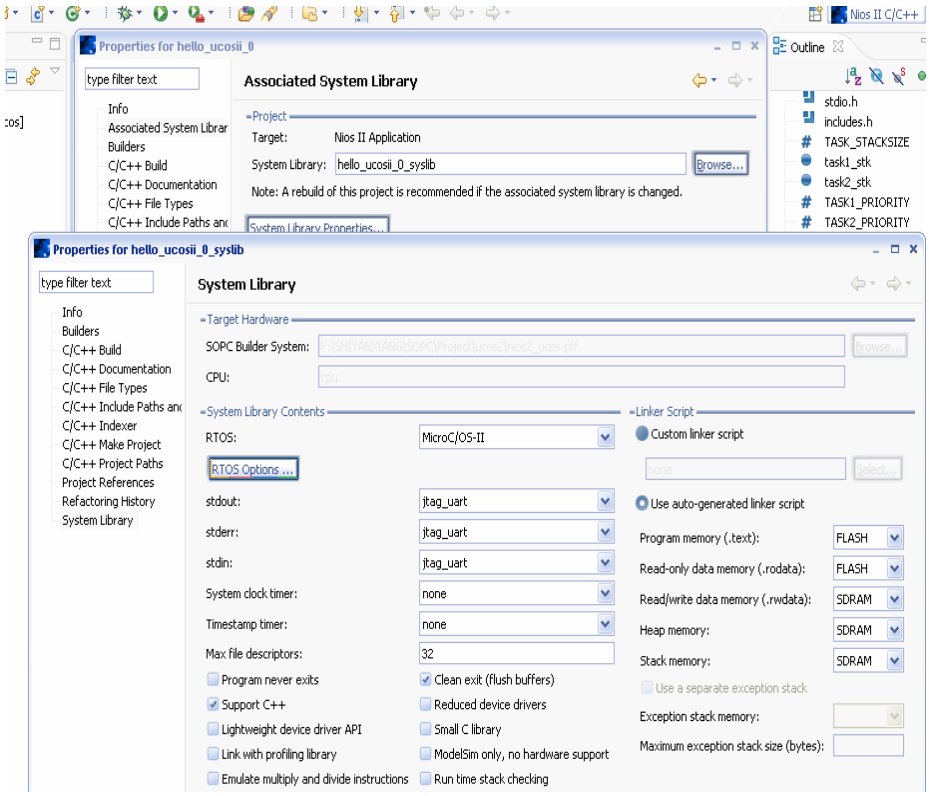


4、启动 Nios2 IDE 。

新建 Nios II C/C++ Aplication 如下图所示：选择 CPU，并选中 Hello MicroC/OS-II 工程实例。



生成工程文件后，设置系统属性。通过 Project→properties 在弹出对话框中选择 System Library Properties 设置如图所示：



5、下载。编译后首先把 Quartus II 中的*.sof 文件下载到 FPGA，然后再由 Nios II IDE 中 Run as---Nios II Hardware. 运行程序。

六、参考程序

```
#include <stdio.h>
```

```
#include "includes.h"

/* Definition of Task Stacks */

#define     TASK_STACKSIZE           2048

OS_STK      task1_stk[TASK_STACKSIZE];

OS_STK      task2_stk[TASK_STACKSIZE]; /* Definition of Task Priorities */

#define TASK1_PRIORITY          7

#define TASK2_PRIORITY          9

/* Prints "Hello World" and sleeps for three seconds */

void task1(void* pdata)

{   while (1)

  {   printf("Hello from task1\n");

      OSTimeDlyHMSM(0, 0, 3, 0);

      inti_UART0();

  }}

/* Prints "Hello World" and sleeps for three seconds */

void task2(void* pdata)

{   while (1)

  {   printf("Hello from task2\n");

      OSTimeDlyHMSM(0, 0, 3, 0);

  }}/* The main function creates two task and starts multi-tasking */

int main(void)

{ OSTaskCreateExt(task1,

                  NULL,

                  (void *)&task1_stk[TASK_STACKSIZE],

                  TASK1_PRIORITY,

                  TASK1_PRIORITY,

                  task1_stk,

                  TASK_STACKSIZE,

                  NULL,

                  0);
```

```
    OSTaskCreateExt(task2,
                    NULL,
                    (void *)&task2_stk[TASK_STACKSIZE],
                    TASK2_PRIORITY,
                    TASK2_PRIORITY,
                    task2_stk,
                    TASK_STACKSIZE,
                    NULL,
                    0);
    OSStart();
    return 0;
}
```

管脚分配：

```
    set_location_assignment PIN_209 -to sys_reset
    set_location_assignment PIN_31 -to sys_clk
    set_location_assignment PIN_146 -to sdram_addr[0]
    set_location_assignment PIN_145 -to sdram_addr[1]
    set_location_assignment PIN_144 -to sdram_addr[2]
    set_location_assignment PIN_111 -to sdram_addr[3]
    set_location_assignment PIN_110 -to sdram_addr[4]
    set_location_assignment PIN_109 -to sdram_addr[5]
    set_location_assignment PIN_108 -to sdram_addr[6]
    set_location_assignment PIN_107 -to sdram_addr[7]
    set_location_assignment PIN_106 -to sdram_addr[8]
    set_location_assignment PIN_103 -to sdram_addr[9]
    set_location_assignment PIN_147 -to sdram_addr[10]
    set_location_assignment PIN_160 -to sdram_addr[11]
    set_location_assignment PIN_159 -to sdram_ba[0]
    set_location_assignment PIN_148 -to sdram_ba[1]
```

```
set_location_assignment PIN_164 -to sdram_cas
set_location_assignment PIN_102 -to sdram_cke
set_location_assignment PIN_117 -to sdram_clk
set_location_assignment PIN_161 -to sdram_cs
set_location_assignment PIN_177 -to sdram_dq[0]
set_location_assignment PIN_176 -to sdram_dq[1]
set_location_assignment PIN_175 -to sdram_dq[2]
set_location_assignment PIN_174 -to sdram_dq[3]
set_location_assignment PIN_173 -to sdram_dq[4]
set_location_assignment PIN_171 -to sdram_dq[5]
set_location_assignment PIN_169 -to sdram_dq[6]
set_location_assignment PIN_168 -to sdram_dq[7]
set_location_assignment PIN_100 -to sdram_dq[8]
set_location_assignment PIN_99 -to sdram_dq[9]
set_location_assignment PIN_98 -to sdram_dq[10]
set_location_assignment PIN_95 -to sdram_dq[11]
set_location_assignment PIN_94 -to sdram_dq[12]
set_location_assignment PIN_93 -to sdram_dq[13]
set_location_assignment PIN_88 -to sdram_dq[14]
set_location_assignment PIN_87 -to sdram_dq[15]
set_location_assignment PIN_142 -to sdram_dq[16]
set_location_assignment PIN_139 -to sdram_dq[17]
set_location_assignment PIN_137 -to sdram_dq[18]
set_location_assignment PIN_135 -to sdram_dq[19]
set_location_assignment PIN_134 -to sdram_dq[20]
set_location_assignment PIN_133 -to sdram_dq[21]
set_location_assignment PIN_132 -to sdram_dq[22]
set_location_assignment PIN_131 -to sdram_dq[23]
set_location_assignment PIN_128 -to sdram_dq[24]
set_location_assignment PIN_127 -to sdram_dq[25]
```

set_location_assignment PIN_126 -to sdram_dq[26]

set_location_assignment PIN_120 -to sdram_dq[27]

set_location_assignment PIN_119 -to sdram_dq[28]

set_location_assignment PIN_118 -to sdram_dq[29]

set_location_assignment PIN_114 -to sdram_dq[30]

set_location_assignment PIN_113 -to sdram_dq[31]

set_location_assignment PIN_167 -to sdram_dqm[0]

set_location_assignment PIN_101 -to sdram_dqm[1]

set_location_assignment PIN_143 -to sdram_dqm[2]

set_location_assignment PIN_162 -to sdram_ras

set_location_assignment PIN_112 -to sdram_dqm[3]

set_location_assignment PIN_166 -to sdram_we

set_location_assignment PIN_181 -to rxd

set_location_assignment PIN_183 -to txd

set_location_assignment PIN_49 -to flash_addr[0]

set_location_assignment PIN_76 -to flash_addr[1]

set_location_assignment PIN_73 -to flash_addr[2]

set_location_assignment PIN_72 -to flash_addr[3]

set_location_assignment PIN_71 -to flash_addr[4]

set_location_assignment PIN_70 -to flash_addr[5]

set_location_assignment PIN_69 -to flash_addr[6]

set_location_assignment PIN_68 -to flash_addr[7]

set_location_assignment PIN_65 -to flash_addr[8]

set_location_assignment PIN_64 -to flash_addr[9]

set_location_assignment PIN_63 -to flash_addr[10]

set_location_assignment PIN_57 -to flash_addr[11]

set_location_assignment PIN_55 -to flash_addr[12]

set_location_assignment PIN_52 -to flash_addr[13]

set_location_assignment PIN_51 -to flash_addr[14]

set_location_assignment PIN_78 -to flash_addr[15]

```
set_location_assignment PIN_80 -to flash_addr[16]
set_location_assignment PIN_81 -to flash_addr[17]
set_location_assignment PIN_82 -to flash_addr[18]
set_location_assignment PIN_83 -to flash_addr[19]
set_location_assignment PIN_84 -to flash_addr[20]
set_location_assignment PIN_85 -to flash_addr[21]
set_location_assignment PIN_56 -to flash_select
set_location_assignment PIN_46 -to flash_data[0]
set_location_assignment PIN_45 -to flash_data[1]
set_location_assignment PIN_44 -to flash_data[2]
set_location_assignment PIN_43 -to flash_data[3]
set_location_assignment PIN_41 -to flash_data[4]
set_location_assignment PIN_39 -to flash_data[5]
set_location_assignment PIN_38 -to flash_data[6]
set_location_assignment PIN_37 -to flash_data[7]
set_location_assignment PIN_22 -to flash_read
set_location_assignment PIN_21 -to flash_write
```

实验十一 PS2 鼠标控制实验

一、实验目的

1、了解 PS2 鼠标键盘协议；

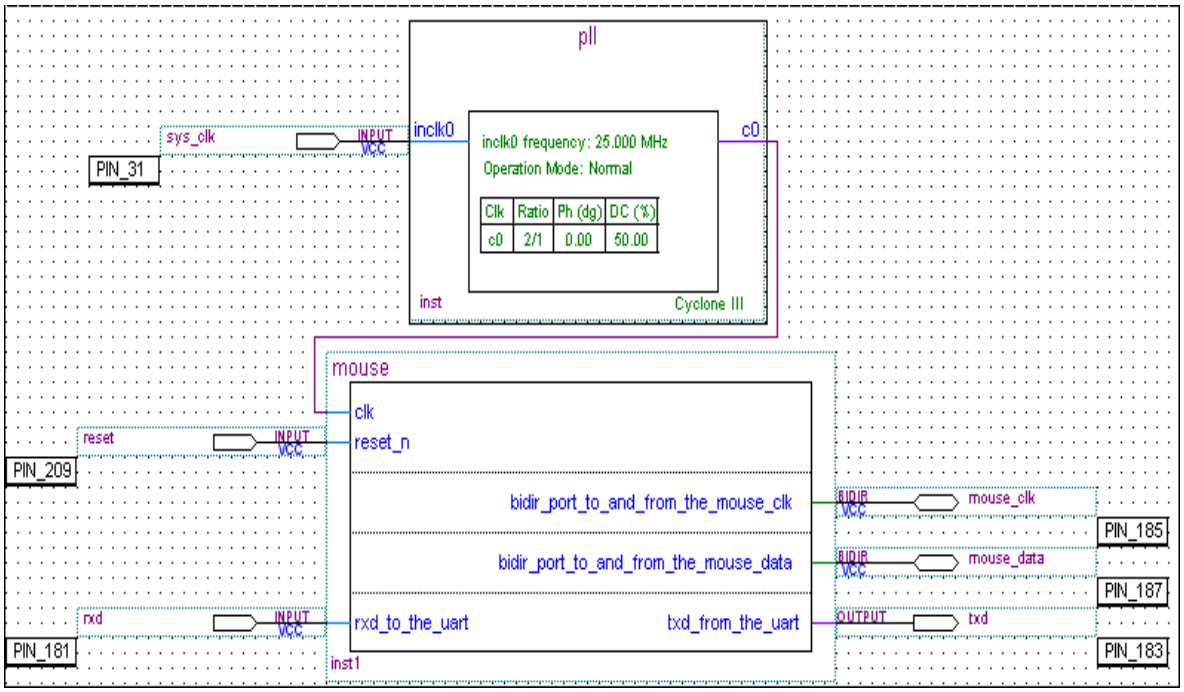
2、学会分析简单的数字信号和使用 Nios II 软和捕捉及及解码信号。

二、实验原理

PS2 接口用于许多现代的鼠标和键盘，由 IBM 最初开发和使用。物理上的 PS2 有两种类型的连接器：5 脚的 DIN 和 6 脚的 MINI-DIN。PS2 鼠标接口采用一种双向同步串行协议，即每在时钟线上发一个脉冲，就在数据线上发送一位数据。在相互传输中，主机拥有总线控制权，即它可以在任何时候抑制鼠标的发送。方法是把时钟线一直拉低，鼠标就不能产生时钟信号和发送数据。在两个方向的传输中，时钟信号都是由鼠标产生，即主机不产生通信时钟信号。

三、实验步骤

1、建立 Nios II 工程，添加两个三态输入输出 IO 口，分别作为鼠标的时钟脚和数据管脚。添加 UART 模块便于调试。生成软核，编译，如下图所示。



2、首先鼠标上电后自检，向主机发送 0xaa、0x00；在这里 FPGA 不进行此项检测，直接发送使能鼠标命令 0xF4，鼠标在接收该命令后以 0xFA 应答，点击或移动鼠标，观察实验效果。

四、参考程序

以下是发送命令部分程序：

```
void Send_To_Mouse(unsigned char command,unsigned char pir)
 {
```

```
 unsigned char i;
 alt_irq_context  status;
 status=alt_irq_disable_all();
 OUT_MOUSE_CLK;
 OUT_MOUSE_DAT;
 CLR_MOUSE_CLK;              //拉低时钟线
 delay_10us(2000);           //拉低至少100us
 CLR_MOUSE_DAT;              //拉低数据线
 SET_MOUSE_CLK;              //拉高时钟线
 IN_MOUSE_CLK;               //释放时钟线
 while(RD_MOUSE_CLK==0X01)CLR_MOUSE_DAT;;
 for(i=0;i<=7;i++)
 {
  while(RD_MOUSE_CLK==0X00);
  {
   if(((command>>i)&0x01)==0x01)
    SET_MOUSE_DAT;          //命令
   else
    CLR_MOUSE_DAT;
  }
  while(RD_MOUSE_CLK==0X01);
 }
 while(RD_MOUSE_CLK==0X00);
 {
  if(pir==0x1)
    SET_MOUSE_DAT;
  else
    CLR_MOUSE_DAT;
 }                           //发送奇校验位
 while(RD_MOUSE_CLK==0X01);
 while(RD_MOUSE_CLK==0X00);
   SET_MOUSE_DAT;            //发送停止位
 while(RD_MOUSE_CLK==0X01);
 while(RD_MOUSE_CLK==0X01);
  IN_MOUSE_DAT;              //接收应答位
 while(RD_MOUSE_CLK==0X00);
 alt_irq_enable_all(status);
}
```

五、实验要求

有余力的同学可以试着将鼠标的坐标信息以坐标点的新式显示到 VGA 显示器上。或参考 PS2 有关资料，自行使用 VHDL 语言描述，PS2 时序。

管脚分配：

set_location_assignment PIN_6 -to rxd

set_location_assignment PIN_4 -to txd

set_location_assignment PIN_31 -to sys_clk

set_location_assignment PIN_209 -to reset

set_location_assignment PIN_19 -to mouse_clk

set_location_assignment PIN_13 -to mouse_data

set_location_assignment PIN_32 -to clk

实验十二　音频接口实验

一、实验目的

1、进一步熟练使用 SOPC 软件及其配置方法，理解操作系统工作原理；

2、掌握其在 NIOS II IDE 中的使用方法；

3、了解数字音频发生及合成原理。

二、实验模块

音频实验模块、FPGA 主控制板模块；

三、实验原理

在 Nios II IDE 中使用 MicroC/OS-II RTOS ，开发人员可以根据具体实验任务的需要通过在面板选项中的设定来自己剪裁程序，可以通过 RTOS Option 选项来自己定义所需要的功能选项。通过多任务协调来处理音频信号。

四、实验内容

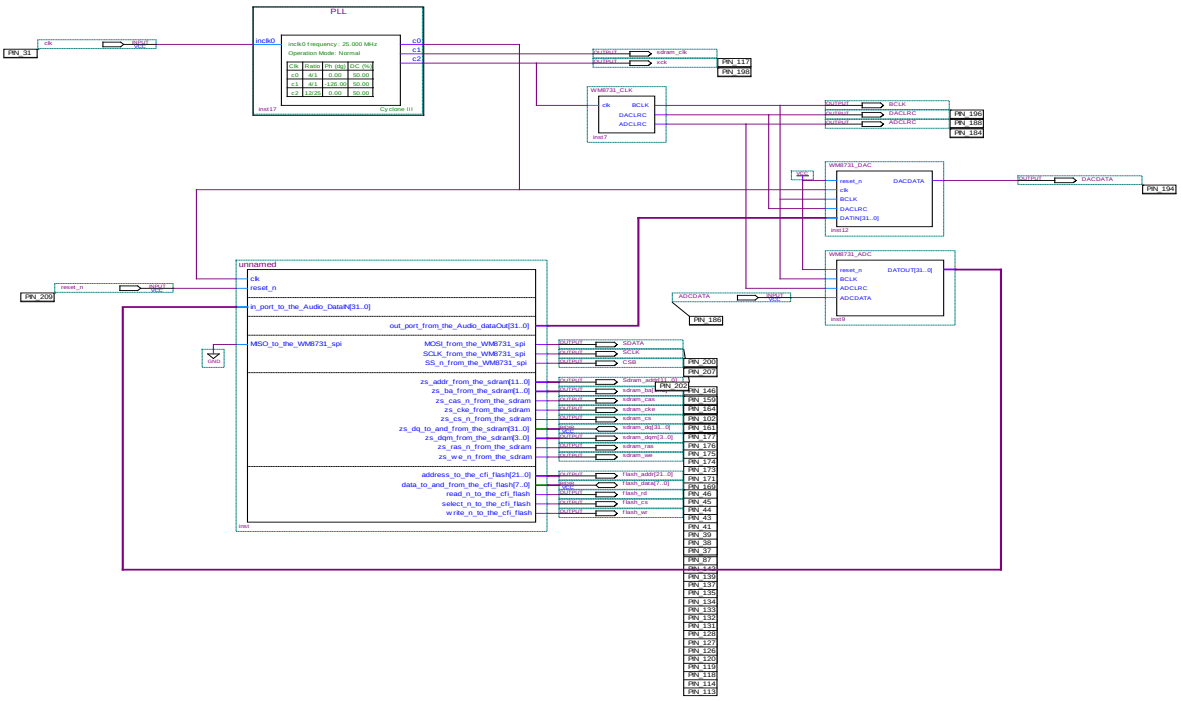
首先使用接入音频信号经音频实验模块转换为数字信号，再由 FPGA 将音频数据发送到 PC 机。

五、实验步骤

1、利用 Quartus II 软件建立工程命名为 Audio，并选择 Cyclone III 器件 EP3C16Q240C8。

2、建立 SOPC Builder，命名为 Nios2_Audio，加入 I/O，SPI，timer，timer_1，sdram，systemID，ram，并生成 CPU。

3、新建原理图文件，添加分配管脚，并编译文件。



4、启动 Nios II IDE 并重新设置工作目录路径，建立新的 C/C++应用程序；选择 Nios2_IDE 的 CPU，按照 UC\OS-II 操作系统移植实验生成工程。生成工程文件后，设置系统属性。

5、下载。编译后首先把 Quartus II 中的*.sof 文件下载到 FPGA。然后再由 Nios II IDE 中 Run as---Nios II Hardware，运行程序，通过控制台控制录音与播放。

六、参考程序

主函数：

```
#include <stdio.h>
#include "includes.h"
#include "altera_avalon_spi_regs.h"
#include "altera_avalon_pio_regs.h"
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#include "wm7137.h"
#include "sys/alt_irq.h"
#include "altera_avalon_timer_regs.h"
#include "altera_avalon_timer.h"
/* Definition of Task Stacks */
#define   TASK_STACKSIZE      2048
OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];
/* Definition of Task Priorities */
#define TASK1_PRIORITY     1
#define TASK2_PRIORITY     2
int TimerFlag;
alt_u32 AudioData[1000000];
alt_u32 DataIndex=0,DataLenth;
alt_u8 stateFlag;
static void Timer_Interrupts(void * context,alt_u32 id)
{
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE, 0);
    if(TimerFlag) TimerFlag=0;
    else
       TimerFlag=0xff;
    if(stateFlag==1)
    {
    AudioData[DataIndex++]=IORD_ALTERA_AVALON_PIO_DATA(AUDIO_DATAIN_BASE);
    }
    if (stateFlag==2)
    {
```

```
IOWR_ALTERA_AVALON_PIO_DATA(AUDIO_DATAOUT_BASE,AudioData[DataIndex++]);
    }
}
void Timer_Initialize(alt_u32 TimerPeriod)
{
       alt_irq_register(TIMER_IRQ ,0,Timer_Interrupts);
       IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_BASE,TimerPeriod &
0x0000ffff);
       IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_BASE,(TimerPeriod >> 16) &
0x0000ffff);
       IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE,
                ALTERA_AVALON_TIMER_CONTROL_START_MSK
                |ALTERA_AVALON_TIMER_CONTROL_ITO_MSK
                |ALTERA_AVALON_TIMER_CONTROL_CONT_MSK);
}
/* Prints "Hello World" and sleeps for three seconds */
void task1(void* pdata)
{
   char cmd[20];
   WM817_Init();
   printf("系统开始\n");
   Timer_Initialize(12500-1);
  while (1)
  {
    scanf("%s",cmd);
    printf("%s",cmd);
    if(strcmp(cmd,"start")==0)
    {
        stateFlag=1;
        printf("录音开始\n");
        DataIndex=0;
    }
    if(strcmp(cmd,"stop")==0)
    {
        stateFlag=0;
        DataLenth=DataIndex;
        printf("录音结束!,共录 %d 秒\n",DataIndex/8000);
    }
    if(strcmp(cmd,"play")==0)
    {
        stateFlag=2;
        DataLenth=DataIndex;
        printf("开始播放\n",DataIndex/8000);
        DataIndex=0;
```

```
    }
  }
}
/* Prints "Hello World" and sleeps for three seconds */
void task2(void* pdata)
{
  while(1)
  {
      if(stateFlag ==1 )
      {

         if(DataIndex>=999999)
         {
            stateFlag=0;
            DataLenth=DataIndex;
            printf("时间到,录音结束!\n");
         }
         printf("正在录音: %d秒\n",DataIndex/8000);
      }
       if(stateFlag ==2 )
      {
          printf("正在回放: %d秒\n",DataIndex/8000);
         if(DataIndex>=DataLenth)
         {
            stateFlag=0;
            printf("时间到,播放结束!\n");
         }
      }
    OSTimeDlyHMSM(0, 0, 1, 0);
  }
}
/* The main function creates two task and starts multi-tasking */
int main(void)
{
  OSTaskCreateExt(task1,
                  NULL,
                  (void *)&task1_stk[TASK_STACKSIZE],
                  TASK1_PRIORITY,
                  TASK1_PRIORITY,
                  task1_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);
```

```
  OSTaskCreateExt(task2,
                  NULL,
                  (void *)&task2_stk[TASK_STACKSIZE],
                  TASK2_PRIORITY,
                  TASK2_PRIORITY,
                  task2_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);
  OSStart();
  return 0;
}
```

管脚分配：

set_location_assignment PIN_31 -to clk

set_location_assignment PIN_49 -to flash_addr[0]

set_location_assignment PIN_76 -to flash_addr[1]

set_location_assignment PIN_73 -to flash_addr[2]

set_location_assignment PIN_72 -to flash_addr[3]

set_location_assignment PIN_71 -to flash_addr[4]

set_location_assignment PIN_70 -to flash_addr[5]

set_location_assignment PIN_69 -to flash_addr[6]

set_location_assignment PIN_68 -to flash_addr[7]

set_location_assignment PIN_65 -to flash_addr[8]

set_location_assignment PIN_64 -to flash_addr[9]

set_location_assignment PIN_63 -to flash_addr[10]

set_location_assignment PIN_57 -to flash_addr[11]

set_location_assignment PIN_55 -to flash_addr[12]

set_location_assignment PIN_52 -to flash_addr[13]

set_location_assignment PIN_51 -to flash_addr[14]

set_location_assignment PIN_78 -to flash_addr[15]

set_location_assignment PIN_80 -to flash_addr[16]

set_location_assignment PIN_81 -to flash_addr[17]

set_location_assignment PIN_82 -to flash_addr[18]

set_location_assignment PIN_83 -to flash_addr[19]

set_location_assignment PIN_84 -to flash_addr[20]

set_location_assignment PIN_85 -to flash_addr[21]

set_location_assignment PIN_56 -to flash_cs

set_location_assignment PIN_46 -to flash_data[0]

set_location_assignment PIN_45 -to flash_data[1]

set_location_assignment PIN_44 -to flash_data[2]

set_location_assignment PIN_43 -to flash_data[3]

set_location_assignment PIN_41 -to flash_data[4]

set_location_assignment PIN_39 -to flash_data[5]

set_location_assignment PIN_38 -to flash_data[6]

set_location_assignment PIN_37 -to flash_data[7]

set_location_assignment PIN_22 -to flash_rd

set_location_assignment PIN_21 -to flash_wr

set_location_assignment PIN_209 -to reset_n

set_location_assignment PIN_146 -to Sdram_addr[0]

set_location_assignment PIN_145 -to Sdram_addr[1]

set_location_assignment PIN_144 -to Sdram_addr[2]

set_location_assignment PIN_111 -to Sdram_addr[3]

set_location_assignment PIN_110 -to Sdram_addr[4]

set_location_assignment PIN_109 -to Sdram_addr[5]

set_location_assignment PIN_108 -to Sdram_addr[6]

set_location_assignment PIN_107 -to Sdram_addr[7]

set_location_assignment PIN_106 -to Sdram_addr[8]

set_location_assignment PIN_103 -to Sdram_addr[9]

set_location_assignment PIN_147 -to Sdram_addr[10]

set_location_assignment PIN_160 -to Sdram_addr[11]

set_location_assignment PIN_159 -to sdram_ba[0]

set_location_assignment PIN_148 -to sdram_ba[1]

set_location_assignment PIN_164 -to sdram_cas

set_location_assignment PIN_102 -to sdram_cke

set_location_assignment PIN_117 -to sdram_clk

set_location_assignment PIN_161 -to sdram_cs

set_location_assignment PIN_177 -to sdram_dq[0]

set_location_assignment PIN_176 -to sdram_dq[1]

set_location_assignment PIN_175 -to sdram_dq[2]

set_location_assignment PIN_174 -to sdram_dq[3]

set_location_assignment PIN_173 -to sdram_dq[4]

set_location_assignment PIN_171 -to sdram_dq[5]

set_location_assignment PIN_169 -to sdram_dq[6]

set_location_assignment PIN_168 -to sdram_dq[7]

set_location_assignment PIN_100 -to sdram_dq[8]

set_location_assignment PIN_99 -to sdram_dq[9]

set_location_assignment PIN_98 -to sdram_dq[10]

set_location_assignment PIN_95 -to sdram_dq[11]

set_location_assignment PIN_94 -to sdram_dq[12]

set_location_assignment PIN_93 -to sdram_dq[13]

set_location_assignment PIN_88 -to sdram_dq[14]

set_location_assignment PIN_87 -to sdram_dq[15]

set_location_assignment PIN_142 -to sdram_dq[16]

set_location_assignment PIN_139 -to sdram_dq[17]

set_location_assignment PIN_137 -to sdram_dq[18]

set_location_assignment PIN_135 -to sdram_dq[19]

set_location_assignment PIN_134 -to sdram_dq[20]

set_location_assignment PIN_133 -to sdram_dq[21]

set_location_assignment PIN_132 -to sdram_dq[22]

set_location_assignment PIN_131 -to sdram_dq[23]

set_location_assignment PIN_128 -to sdram_dq[24]

set_location_assignment PIN_127 -to sdram_dq[25]

set_location_assignment PIN_126 -to sdram_dq[26]

```
set_location_assignment PIN_120 -to sdram_dq[27]
set_location_assignment PIN_119 -to sdram_dq[28]
set_location_assignment PIN_118 -to sdram_dq[29]
set_location_assignment PIN_114 -to sdram_dq[30]
set_location_assignment PIN_113 -to sdram_dq[31]
set_location_assignment PIN_167 -to sdram_dqm[0]
set_location_assignment PIN_101 -to sdram_dqm[1]
set_location_assignment PIN_143 -to sdram_dqm[2]
set_location_assignment PIN_112 -to sdram_dqm[3]
set_location_assignment PIN_162 -to sdram_ras
set_location_assignment PIN_166 -to sdram_we
set_location_assignment PIN_186 -to ADCDATA
set_location_assignment PIN_184 -to ADCLRC
set_location_assignment PIN_196 -to BCLK
set_location_assignment PIN_207 -to CSB
set_location_assignment PIN_194 -to DACDATA
set_location_assignment PIN_188 -to DACLRC
set_location_assignment PIN_200 -to SCLK
set_location_assignment PIN_202 -to SDATA
set_location_assignment PIN_198 -to xck
```

实验十三　百兆以太网实验

一、实验目的

熟练使用 SOPC 软件，掌握操作系统工作原理，理解以太网数据传输的原理及其数据报的组成，掌握以太网在 IP 核的使用方法。

二、实验模块

10/100M 以太网通信模块、FPGA 主控制板模块。

三、实验原理

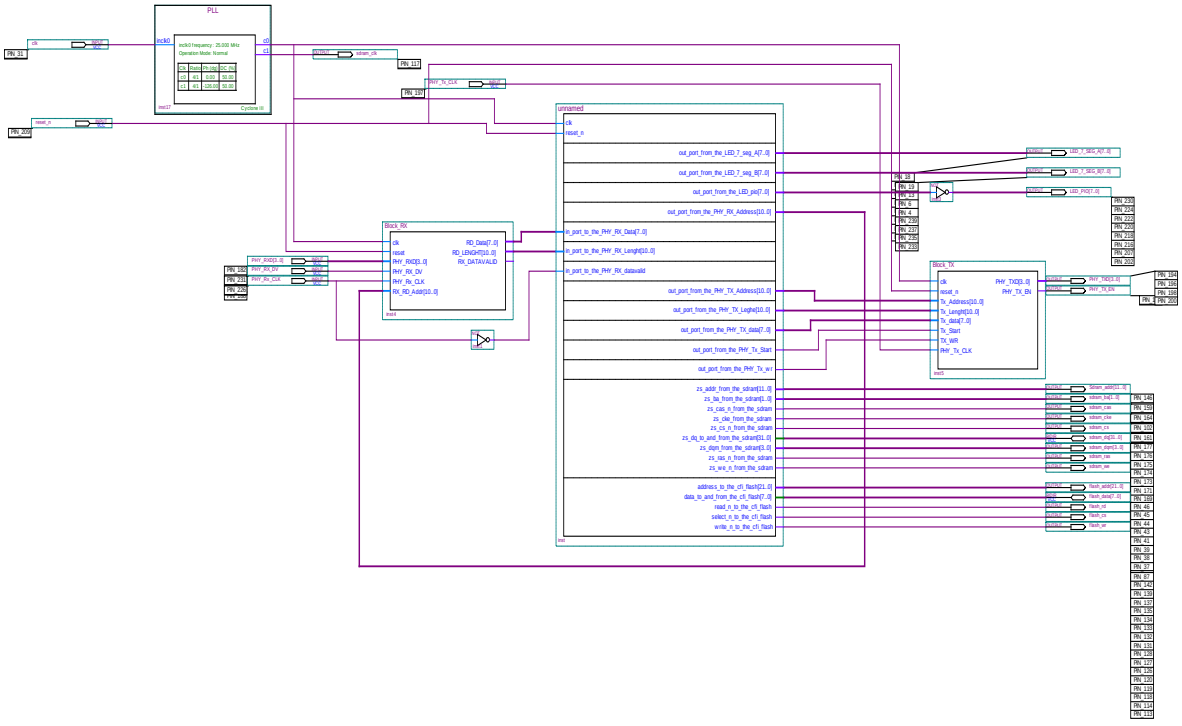
实验运用 DP83848 以太网管理芯片来收发数据，并通过上位机控制界面来发送控制命令，控制数码管和 LED 灯的亮灭。

四、实验步骤

1、利用 Quartus II 软件建立工程命名为 PHYTest，并选择 Cyclone III 器件 EP3C16Q240C8。

2、建立 SOPC Builder，加入 I/O 口，JTAG，systemID，timer，sdram 等，建立 10k 以上的 RAM 空间，并生成 CPU。

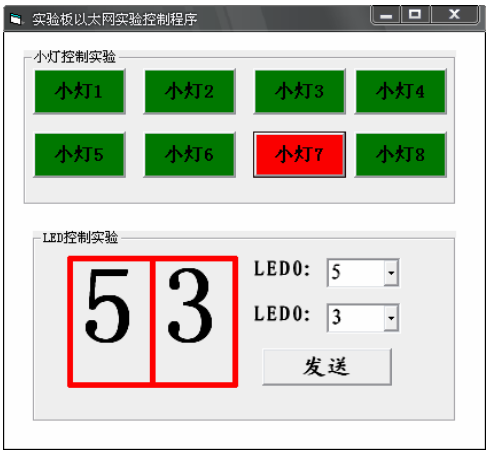
3、新建原理图文件，新建 Block_Rx 和 Block_TxVHDL 文件，并生成模块；搭建硬件图，添加分配管脚，并编译文件。



4、启动 Nios II IDE 并重新设置工作目录路径，建立新的 C/C++应用程序；选择 Nios2_IDE 的 CPU 和示例“hello word”，生成工程。生成工程文件后，设置系统属性。通过 Project→properties 在弹出对话框中选择 System Library Properties 设置同上实验。

5、下载。编译后首先把 Quartus II 中的*.sof 文件下载到 FPGA 然后再由 Nios II IDE 中 Run as---Nios II Hardware. 运行程序。

6、通过 VB 编写的上位机控制界面来实现小灯的亮灭以及数码管的数字显示。



五、参考程序

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
static alt_u8 RX_buff[2000];
static alt_u16 RX_lengh;
static int RX_FLAG=0;
static alt_u8
LED_CODE[17]={0xc0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83
,0xc6,0xa1,0x86,0x8e,0x8c,0xc1,0xce,0x91,0x00,0xff};
int main(void)
{
  int i;
  char str[10];
  alt_u8 led=0;
    IOWR_ALTERA_AVALON_PIO_DATA(LED_7_SEG_A_BASE,LED_CODE[0]);
    IOWR_ALTERA_AVALON_PIO_DATA(LED_7_SEG_B_BASE,LED_CODE[0]);
    printf("Hello from Nios II!\n");
    RX_lengh=0;
    while (1)
   {
    if(!IORD_ALTERA_AVALON_PIO_DATA(PHY_RX_DATAVALID_BASE))
        RX_FLAG=1;
    if(RX_FLAG==1)
    {
```

```
        RX_FLAG=0;
        RX_lengh=IORD_ALTERA_AVALON_PIO_DATA(PHY_RX_LENGHT_BASE);
        for(i=0;i<RX_lengh;i++)
        {
           IOWR_ALTERA_AVALON_PIO_DATA(PHY_RX_ADDRESS_BASE,i);
           RX_buff[i]=(IORD_ALTERA_AVALON_PIO_DATA(PHY_RX_DATA_BASE))&0xff;
         }
         printf("Receive data lenght=%d",RX_lengh);
         for(i=0;i<RX_lengh;i++)
        {
           if(i%4==0) printf("\n");
           printf("%02x ",RX_buff[i]);
         }
        memset(str,0,10);
        str[0]=RX_buff[50];
        str[1]=RX_buff[51];
        str[2]=RX_buff[52];
        str[3]=RX_buff[53];
        str[4]=RX_buff[54];
        printf("%s \n",str);
        if(str[0]=='L' && str[1]=='E' && str[2]=='D')
        {
            if(str[4]=='0')
               led&=~(1<<(str[3]-'0'));
            else
               led|=(1<<(str[3]-'0'));
            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE,led);
        }
        if(str[0]=='L' && str[1]=='C' && str[2]=='D')
        {
 IOWR_ALTERA_AVALON_PIO_DATA(LED_7_SEG_A_BASE,LED_CODE[(str[3]-'0')]);
 IOWR_ALTERA_AVALON_PIO_DATA(LED_7_SEG_B_BASE,LED_CODE[(str[4]-'0')]);
        }
    }
  }
  return 0;
}
```

管脚分配：

set_location_assignment PIN_31 -to clk

set_location_assignment PIN_49 -to flash_addr[0]

set_location_assignment PIN_76 -to flash_addr[1]

set_location_assignment PIN_73 -to flash_addr[2]

set_location_assignment PIN_72 -to flash_addr[3]

set_location_assignment PIN_71 -to flash_addr[4]

set_location_assignment PIN_70 -to flash_addr[5]

set_location_assignment PIN_69 -to flash_addr[6]

set_location_assignment PIN_68 -to flash_addr[7]

set_location_assignment PIN_65 -to flash_addr[8]

set_location_assignment PIN_64 -to flash_addr[9]

set_location_assignment PIN_63 -to flash_addr[10]

set_location_assignment PIN_57 -to flash_addr[11]

set_location_assignment PIN_55 -to flash_addr[12]

set_location_assignment PIN_52 -to flash_addr[13]

set_location_assignment PIN_51 -to flash_addr[14]

set_location_assignment PIN_78 -to flash_addr[15]

set_location_assignment PIN_80 -to flash_addr[16]

set_location_assignment PIN_81 -to flash_addr[17]

set_location_assignment PIN_82 -to flash_addr[18]

set_location_assignment PIN_83 -to flash_addr[19]

set_location_assignment PIN_84 -to flash_addr[20]

set_location_assignment PIN_85 -to flash_addr[21]

set_location_assignment PIN_56 -to flash_cs

set_location_assignment PIN_46 -to flash_data[0]

set_location_assignment PIN_45 -to flash_data[1]

set_location_assignment PIN_44 -to flash_data[2]

set_location_assignment PIN_43 -to flash_data[3]

set_location_assignment PIN_41 -to flash_data[4]

set_location_assignment PIN_39 -to flash_data[5]

set_location_assignment PIN_38 -to flash_data[6]

set_location_assignment PIN_37 -to flash_data[7]

set_location_assignment PIN_22 -to flash_rd

```
set_location_assignment PIN_21 -to flash_wr
set_location_assignment PIN_230 -to LED_PIO[0]
set_location_assignment PIN_224 -to LED_PIO[1]
set_location_assignment PIN_222 -to LED_PIO[2]
set_location_assignment PIN_220 -to LED_PIO[3]
set_location_assignment PIN_218 -to LED_PIO[4]
set_location_assignment PIN_216 -to LED_PIO[5]
set_location_assignment PIN_207 -to LED_PIO[6]
set_location_assignment PIN_202 -to LED_PIO[7]
set_location_assignment PIN_209 -to reset_n
set_location_assignment PIN_146 -to Sdram_addr[0]
set_location_assignment PIN_145 -to Sdram_addr[1]
set_location_assignment PIN_144 -to Sdram_addr[2]
set_location_assignment PIN_111 -to Sdram_addr[3]
set_location_assignment PIN_110 -to Sdram_addr[4]
set_location_assignment PIN_109 -to Sdram_addr[5]
set_location_assignment PIN_108 -to Sdram_addr[6]
set_location_assignment PIN_107 -to Sdram_addr[7]
set_location_assignment PIN_106 -to Sdram_addr[8]
set_location_assignment PIN_103 -to Sdram_addr[9]
set_location_assignment PIN_147 -to Sdram_addr[10]
set_location_assignment PIN_160 -to Sdram_addr[11]
set_location_assignment PIN_159 -to sdram_ba[0]
set_location_assignment PIN_148 -to sdram_ba[1]
set_location_assignment PIN_164 -to sdram_cas
set_location_assignment PIN_102 -to sdram_cke
set_location_assignment PIN_117 -to sdram_clk
set_location_assignment PIN_161 -to sdram_cs
set_location_assignment PIN_177 -to sdram_dq[0]
set_location_assignment PIN_176 -to sdram_dq[1]
```

set_location_assignment PIN_175 -to sdram_dq[2]

set_location_assignment PIN_174 -to sdram_dq[3]

set_location_assignment PIN_173 -to sdram_dq[4]

set_location_assignment PIN_171 -to sdram_dq[5]

set_location_assignment PIN_169 -to sdram_dq[6]

set_location_assignment PIN_168 -to sdram_dq[7]

set_location_assignment PIN_100 -to sdram_dq[8]

set_location_assignment PIN_99 -to sdram_dq[9]

set_location_assignment PIN_98 -to sdram_dq[10]

set_location_assignment PIN_95 -to sdram_dq[11]

set_location_assignment PIN_94 -to sdram_dq[12]

set_location_assignment PIN_93 -to sdram_dq[13]

set_location_assignment PIN_88 -to sdram_dq[14]

set_location_assignment PIN_87 -to sdram_dq[15]

set_location_assignment PIN_142 -to sdram_dq[16]

set_location_assignment PIN_139 -to sdram_dq[17]

set_location_assignment PIN_137 -to sdram_dq[18]

set_location_assignment PIN_135 -to sdram_dq[19]

set_location_assignment PIN_134 -to sdram_dq[20]

set_location_assignment PIN_133 -to sdram_dq[21]

set_location_assignment PIN_132 -to sdram_dq[22]

set_location_assignment PIN_131 -to sdram_dq[23]

set_location_assignment PIN_128 -to sdram_dq[24]

set_location_assignment PIN_127 -to sdram_dq[25]

set_location_assignment PIN_126 -to sdram_dq[26]

set_location_assignment PIN_120 -to sdram_dq[27]

set_location_assignment PIN_119 -to sdram_dq[28]

set_location_assignment PIN_118 -to sdram_dq[29]

set_location_assignment PIN_114 -to sdram_dq[30]

set_location_assignment PIN_113 -to sdram_dq[31]

set_location_assignment PIN_167 -to sdram_dqm[0]

set_location_assignment PIN_101 -to sdram_dqm[1]

set_location_assignment PIN_143 -to sdram_dqm[2]

set_location_assignment PIN_112 -to sdram_dqm[3]

set_location_assignment PIN_162 -to sdram_ras

set_location_assignment PIN_166 -to sdram_we

set_location_assignment PIN_189 -to PHY_MDC

set_location_assignment PIN_195 -to PHY_MDIO

set_location_assignment PIN_185 -to PHY_Rx_CLK

set_location_assignment PIN_187 -to PHY_RX_DV

set_location_assignment PIN_182 -to PHY_RXD[0]

set_location_assignment PIN_184 -to PHY_RXD[1]

set_location_assignment PIN_186 -to PHY_RXD[2]

set_location_assignment PIN_188 -to PHY_RXD[3]

set_location_assignment PIN_181 -to PHY_Tx_CLK

set_location_assignment PIN_183 -to PHY_TX_EN

set_location_assignment PIN_194 -to PHY_TXD[0]

set_location_assignment PIN_196 -to PHY_TXD[1]

set_location_assignment PIN_198 -to PHY_TXD[2]

set_location_assignment PIN_200 -to PHY_TXD[3]

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_trigger_in[0] -to PHY_TXD[0] -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_trigger_in[1] -to PHY_TXD[1] -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_trigger_in[2] -to PHY_TXD[2] -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_trigger_in[3] -to PHY_TXD[3] -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_trigger_in[4] -to PHY_TX_EN -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_data_in[0] -to PHY_TXD[0] -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_data_in[1] -to PHY_TXD[1] -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_data_in[2] -to PHY_TXD[2] -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_data_in[3] -to PHY_TXD[3] -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_data_in[4] -to PHY_TX_EN -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_clk -to sdram_clk -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_trigger_in[5] -to TX_clk_out -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_trigger_in[6] -to TX_start_out -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_trigger_in[7] -to temp_bitflag -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_data_in[5] -to TX_clk_out -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_data_in[6] -to TX_start_out -section_id auto_signaltap_0

set_instance_assignment -name CONNECT_TO_SLD_NODE_ENTITY_PORT acq_data_in[7] -to temp_bitflag -section_id auto_signaltap_0

set_instance_assignment -name POST_FIT_CONNECT_TO_SLD_NODE_ENTITY_PORT acq_trigger_in[8] -to PHY_Tx_CLK -section_id auto_signaltap_0

set_instance_assignment -name POST_FIT_CONNECT_TO_SLD_NODE_ENTITY_PORT acq_data_in[8] -to PHY_Tx_CLK -section_id auto_signaltap_0

set_location_assignment PIN_19 -to temp_bitflag

set_location_assignment PIN_18 -to TX_start_out

```
set_location_assignment PIN_13 -to tx_wr_out
set_location_assignment PIN_18 -to LED_7_SEG_A[0]
set_location_assignment PIN_9 -to LED_7_SEG_A[1]
set_location_assignment PIN_5 -to LED_7_SEG_A[2]
set_location_assignment PIN_240 -to LED_7_SEG_A[3]
set_location_assignment PIN_238 -to LED_7_SEG_A[4]
set_location_assignment PIN_236 -to LED_7_SEG_A[5]
set_location_assignment PIN_234 -to LED_7_SEG_A[6]
set_location_assignment PIN_232 -to LED_7_SEG_A[7]
set_location_assignment PIN_19 -to LED_7_SEG_B[0]
set_location_assignment PIN_13 -to LED_7_SEG_B[1]
set_location_assignment PIN_6 -to LED_7_SEG_B[2]
set_location_assignment PIN_4 -to LED_7_SEG_B[3]
set_location_assignment PIN_239 -to LED_7_SEG_B[4]
set_location_assignment PIN_237 -to LED_7_SEG_B[5]
set_location_assignment PIN_235 -to LED_7_SEG_B[6]
set_location_assignment PIN_233 -to LED_7_SEG_B[7]
```

实验十四　四相步进电机实验

一、实验目的

1．学习步进电机工作原理；

2．学习步进电机与 FPGA 的接口电路设计和编程。

二、实验模块

1．FPGA 系统主模块；

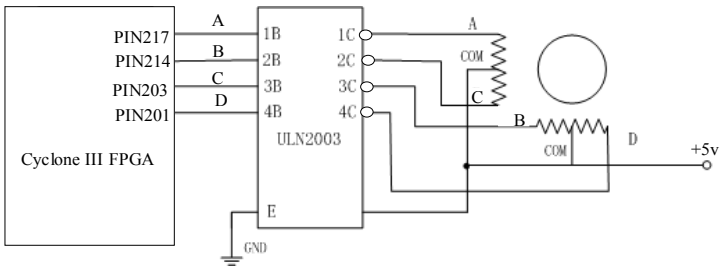
2．步进电机实验模块。

三、实验内容

要求采用 4 相 8 拍的工作方式控制步进电机的正转、反转和增、减转速。

四、实验原理

步进电机是一种将电脉冲信号转换为直线或角位移的执行机构。对其施加1个电脉冲信号后，其转轴就转过一定的角度, 称为1步。对应的角度称为步距角，4相绕组的步进电机其步距角一般为0.9°/1.8°或0.75°/1.5°。输入到步进电机的脉冲数增加, 直线或角位移也随之增加；脉冲频率高, 步进电机的旋转速度就高, 反之则慢；改变加在绕组上的脉冲序列的相序, 电机便逆转。其控制图如下：



步进电机的控制接口电路

实验装置上采用的步进电机为四相 6 线制混合型步进电机 20BY20L01-02A，黄-A，白-B，红-C，黑-D，橙和棕经一定的限流后接＋5VDC。通过 FPGA 口线按顺序给 A、B、C、D 绕相组施加有序的脉冲直流，就可以控制电机的转动，从而完成了数字→角度的转换。转动的角度大小与施加的脉冲数成正比，转动的速度与脉冲频率成正比，而转动方向则与脉冲的顺序有关。

ULN2003 是一个大电流驱动器，为达林顿管阵列电路，可输出 500mA 电流，同时起到电路隔离作用，各输出端与 COM 间有起保护作用的反相二极管。

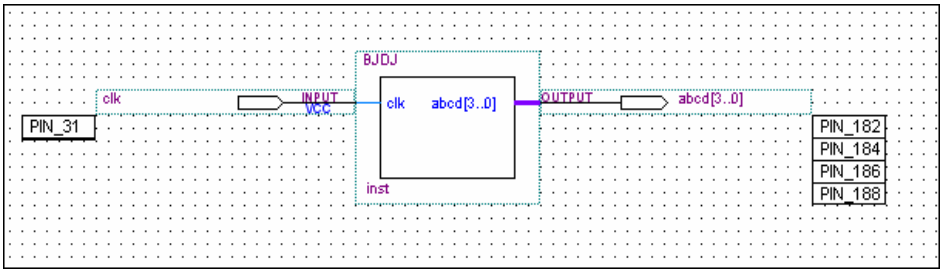
四相步进电机的工作方式：

双 4 拍的工作方式：正转绕组通电顺序为 AB→BC→CD→DA；反转绕组通电顺序为 AD→CD→BC→AB，则应依次给端口送 03H→06H→0CH→09H→03H，反转则 09H→0CH→06H→03H。

五、实验步骤

1、按照上述原理，用导线正确连接步进电机实验模块和 FPGA 系统模块。

2、正确连接 PC 机与目标板。正确连接目标板电源。建立工程并调试，将生成的 SOF 文件下载到 FPGA 即可。程序运行，观察程序运行结果。顶层文件如下所示：



六、参考程序

```
LIBRARY IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity BJDJ is
port(
     clk: in std_logic;
     abcd: out std_logic_vector(3 downto 0)
    );
end BJDJ;
architecture arcount of BJDJ is
shared variable counter :integer :=0;
signal fre_clk : std_logic :='0';
BEGIN
fre_counter:
    PROCESS(clk)
    begin
    if clk'event and clk='1' then
           if(counter<100000) then counter:=counter+1;
              fre_clk<='1';
           elsif(counter<200000)then counter:=counter+1;
              fre_clk<='0';
           else
              counter:=0;
```

```
            end if;
        end if;
        end process;
    foun: process(fre_clk,clk)
      variable cnt:integer :=0;
      begin
      if(fre_clk'event and fre_clk='1') then
              case cnt is
    --        when 1 => abcd<="0001";cnt:=2;
              when 2 => abcd<="0011";cnt:=4;
    --        when 3 => abcd<="0010";cnt:=4;
              when 4 => abcd<="0110";cnt:=6;
    --        when 5 => abcd<="0100";cnt:=6;
              when 6 => abcd<="1100";cnt:=8;
    --        when 7 => abcd<="1000";cnt:=8;
              when 8 => abcd<="1001";cnt:=2;
              when others => abcd<="0000"; cnt:=2;
         end case;
      end if;
     end process;
    end architecture;
```

管脚分配：

```
set_location_assignment PIN_182 -to abcd[3]
set_location_assignment PIN_184 -to abcd[2]
set_location_assignment PIN_186 -to abcd[1]
set_location_assignment PIN_188 -to abcd[0]
set_location_assignment PIN_31 -to clk
```

实验十五 CAN总线实验

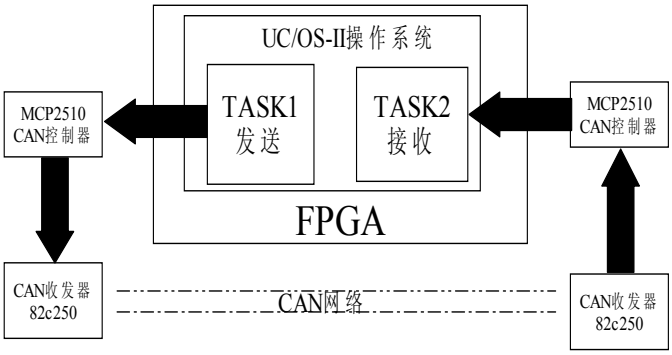
一、 实验要求

1、了解 CAN 总线的通信实验原理；

2、移植 UC/OS-II 利用 MCP2510 实现 CAN 总线上的数据传输 。

二、实验原理

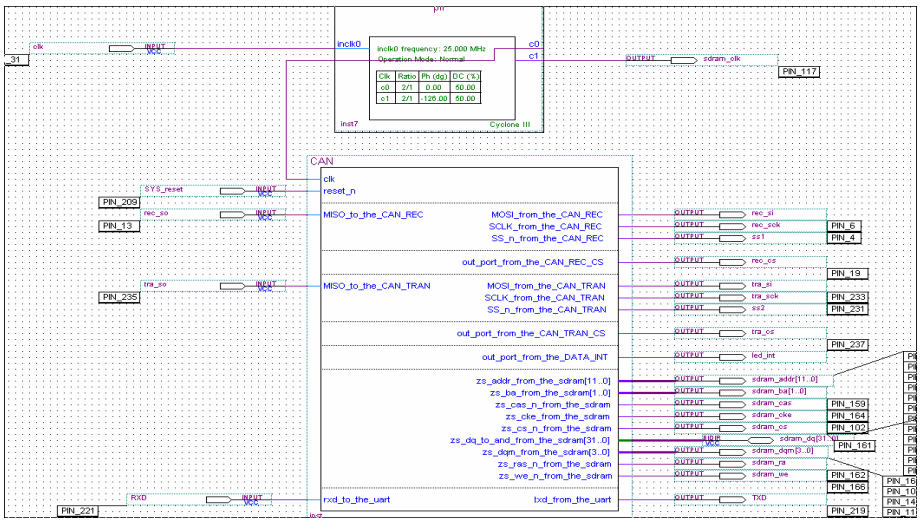
目前市场上的 CAN 控制器芯片主要有两种，最热门的一种是 Philips 公司的 SJA1000，我们选用另一种是成本较低的 Microchip 公司的 MCP2510,它与 FPGA 微控制器的接口只有 4 根线，采用 SPI 接口操作方式，在使用 CAN 控制器 MCP2510 时，需要对 MCP2510 进行初始化以及对 CAN 总线上的数据进行收发操作。详细参考 MCP2510.PDF。实验设计如下图所示：(也可以根据需求自己设计其他实验效果)



三、实验步骤

1、 建立工程命名为 CAN_BUS，并启动 SOPC Builder 命名，添加 jtag_uart，cpu，sdram，以及 spi 口和 PIO 口，最后生成 CPU。

2、 新建原理图文件，加载生成的 CPU，并按照存储器实验设置 SDRAM 的时钟偏移量新建锁相环，然后分配管脚（详细参考实验例程），连线（注意 CANH-CANH，CANL-CANL）。



3、 启动 Nios II IDE，并移植 UC/OS-II 操作系统，重新设置任务优先级，并编写 MCP2510 的驱动程序。本例程使用查询方式接收数据。

四、参考程序

1)、发送程序：

```
void TXMSG(unsigned char CH,int DLC)//发送数据子程序 ，DLC为要发送的数据个数
{
 int i;
 printf("-----发送%d个数据:",DLC);
 for(i=0;i<DLC;i++) printf("%d ",a[i]);
 printf("\r\n");
 printf("\r\n");
 for(i=0;i<DLC;i++)
 {
  WRmcp2510(CH,0x36+i,a[i]);//准备要发送的数据
 }
 WRmcp2510(CH,0x35,DLC);
 WRmcp2510(CH,0x30,0x03);
 WRmcp2510(CH,0x31,RecID_H);
 WRmcp2510(CH,0x32,RecID_L);
 RTS2510(CH,0x01); // 请求发送
 return;
}
```

2)、接收数据程序：

```
int RXMSG(unsigned char CH)// 接收数据子程序
{
 int k,i;
 k=Readmcp2510(CH,0x2c)&0x01;
 if(k==1)
 {
  WRmcp2510(CH,0x2c,Readmcp2510(CH,0x2c)&0xfe);
  RecID_H=Readmcp2510(CH,0x61);
  RecID_L=Readmcp2510(CH,0x62)&0xe0;
  DLC=Readmcp2510(CH,0x65)&0x0f;
  for(i=0;i<DLC;i++){
  c[i]=Readmcp2510(CH,0x66+i);
  }
  printf("-----接收到数据%d个数据:",DLC);
  for(i=0;i<DLC;i++) printf("%d ",c[i]);
  printf("\r\n");
  printf("\r\n");
  return 1;
```

```
  }
  printf("-----总线无数据！\r\n\r\n");
  return 0;
}
```

3)、MCP2510 初始化程序

```
void INIT2510(unsigned char CH)//对MCP2510进行初始化
{
 RESET2510(CH); // 使芯片复位
 SETCONFIG(CH);
 WRmcp2510(CH,0x28,0x02);
 WRmcp2510(CH,0x29,0x90);
 WRmcp2510(CH,0x2a,0x07);
 WRmcp2510(CH,0x21,0x00);
 WRmcp2510(CH,0x22,0x00);
 WRmcp2510(CH,0x00,0x00);
 WRmcp2510(CH,0x01,0x00);
 WRmcp2510(CH,0x2b,0x00);
 SETNORMAL(CH); // 设置为正常操作模式
 return;
}
```

五、结果分析

下载运行程序后，在控制台能够看到发送和接收到的数据。

管脚分配：

set_location_assignment PIN_209 -to SYS_reset

set_location_assignment PIN_146 -to sdram_addr[0]

set_location_assignment PIN_145 -to sdram_addr[1]

set_location_assignment PIN_144 -to sdram_addr[2]

set_location_assignment PIN_111 -to sdram_addr[3]

set_location_assignment PIN_110 -to sdram_addr[4]

set_location_assignment PIN_109 -to sdram_addr[5]

set_location_assignment PIN_108 -to sdram_addr[6]

set_location_assignment PIN_107 -to sdram_addr[7]

set_location_assignment PIN_106 -to sdram_addr[8]

set_location_assignment PIN_103 -to sdram_addr[9]

set_location_assignment PIN_147 -to sdram_addr[10]

set_location_assignment PIN_160 -to sdram_addr[11]

set_location_assignment PIN_159 -to sdram_ba[0]

set_location_assignment PIN_148 -to sdram_ba[1]

set_location_assignment PIN_164 -to sdram_cas

set_location_assignment PIN_102 -to sdram_cke

set_location_assignment PIN_117 -to sdram_clk

set_location_assignment PIN_161 -to sdram_cs

set_location_assignment PIN_177 -to sdram_dq[0]

set_location_assignment PIN_176 -to sdram_dq[1]

set_location_assignment PIN_175 -to sdram_dq[2]

set_location_assignment PIN_174 -to sdram_dq[3]

set_location_assignment PIN_173 -to sdram_dq[4]

set_location_assignment PIN_171 -to sdram_dq[5]

set_location_assignment PIN_169 -to sdram_dq[6]

set_location_assignment PIN_168 -to sdram_dq[7]

set_location_assignment PIN_100 -to sdram_dq[8]

set_location_assignment PIN_99 -to sdram_dq[9]

set_location_assignment PIN_98 -to sdram_dq[10]

set_location_assignment PIN_95 -to sdram_dq[11]

set_location_assignment PIN_94 -to sdram_dq[12]

set_location_assignment PIN_93 -to sdram_dq[13]

set_location_assignment PIN_88 -to sdram_dq[14]

set_location_assignment PIN_87 -to sdram_dq[15]

set_location_assignment PIN_142 -to sdram_dq[16]

set_location_assignment PIN_139 -to sdram_dq[17]

set_location_assignment PIN_137 -to sdram_dq[18]

set_location_assignment PIN_135 -to sdram_dq[19]

set_location_assignment PIN_134 -to sdram_dq[20]

set_location_assignment PIN_133 -to sdram_dq[21]

set_location_assignment PIN_132 -to sdram_dq[22]

```
set_location_assignment PIN_131 -to sdram_dq[23]
set_location_assignment PIN_128 -to sdram_dq[24]
set_location_assignment PIN_127 -to sdram_dq[25]
set_location_assignment PIN_126 -to sdram_dq[26]
set_location_assignment PIN_120 -to sdram_dq[27]
set_location_assignment PIN_119 -to sdram_dq[28]
set_location_assignment PIN_118 -to sdram_dq[29]
set_location_assignment PIN_114 -to sdram_dq[30]
set_location_assignment PIN_113 -to sdram_dq[31]
set_location_assignment PIN_167 -to sdram_dqm[0]
set_location_assignment PIN_101 -to sdram_dqm[1]
set_location_assignment PIN_143 -to sdram_dqm[2]
set_location_assignment PIN_112 -to sdram_dqm[3]
set_location_assignment PIN_166 -to sdram_we
set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -section_id Top
set_location_assignment PIN_31 -to clk
set_location_assignment PIN_19 -to rec_cs
set_location_assignment PIN_4 -to rec_sck
set_location_assignment PIN_13 -to rec_so
set_location_assignment PIN_6 -to rec_si
set_location_assignment PIN_237 -to tra_cs
set_location_assignment PIN_231 -to tra_sck
set_location_assignment PIN_233 -to tra_si
set_location_assignment PIN_235 -to tra_so
set_location_assignment PIN_162 -to sdram_ra
set_location_assignment PIN_221 -to RXD
set_location_assignment PIN_219 -to TXD
```